Vrije
Universiteit
Brussel

Faculty of Science and
Bio-Engineering Sciences
Department of Computer Science

Multi-Criteria Reinforcement Learning for Sequential Decision Making Problems

Dissertation submitted in fulfilment of the requirements for the degree of Doctor of Science: Computer Science

Kristof Van Moffaert

Promotor: Prof. Dr. Ann Nowé

2016 Uitgeverij VUBPRESS Brussels University Press
VUBPRESS is an imprint of ASP nv (Academic and Scientific Publishers nv)
Ravensteingalerij 28
B-1000 Brussels
Tel. +32 (0)2 289 26 50
Fax +32 (0)2 289 26 59
E-mail: info@vubpress.be
www.vubpress.be

ISBN 978 90 5718 094 1
NUR 993
Legal deposit D/2014/11.161/062

Luctor et emergo.

Abstract

Reinforcement learning provides a framework for algorithms that learn from interaction with an unknown environment. Over the years, several researchers have contributed to this framework by proposing algorithms that optimise their behaviour over time. This learning process is characterised by maximising a single, scalar reward signal. However, many real-life problems are inherently too complex to be described by a single, scalar criterion. Usually, these problems involve multiple objectives that often give rise to a conflict of interest. An example can be found in the manufacturing industry where the goal of a company is to come up with a policy that, at the same time, maximises the profit and the user satisfaction while minimising the production costs, the labour costs and the environmental impact. For these types of problems, the traditional reinforcement learning framework is insufficient and lacks expressive power.

In this dissertation, we argue the need for particular reinforcement learning algorithms that are specifically tailored for multi-objective problems. These algorithms are multi-objective reinforcement learning (MORL) algorithms that provide one or more Pareto optimal balances of the problem's original objectives. What this balance intrinsically conveys depends on the emphasis and the preferences of the decision maker. In the case the decision maker's preferences are clear and known a priori, single-policy techniques such as scalarisation functions can be employed to guide the search towards a particular compromise solution. In this light, we analyse several instantiations of scalarisation functions such as the linear and Chebyshev scalarisation function and we demonstrate how they influence the converged solution.

In case the preference articulation of the decision maker is unclear before the optimisation process takes place, it might be appropriate to provide a set of Pareto optimal trade-off solutions to the decision maker that each comprise a different balance of the objectives. One possibility to obtain a set of solutions is to collect the outcomes of several scalarisation functions while varying their direction of search. In this dissertation, we propose two algorithms that follow this principle for both discrete and continuous Pareto fronts.

A more advanced idea we pursue as well is to learn a set of compromise solutions in a simultaneous fashion. In order to accommodate for a genuine multi-policy algorithm, we extend the core principles of the reinforcement learning framework, including the knowledge representation and the bootstrapping process. For continuous action spaces, we propose the multi-objective hierarchical optimistic optimisation algorithm that incrementally constructs a binary tree of optimistic vectorial estimates. For sequential decision making problems, we introduce Pareto Q-learning. Pareto Q-learning is the first temporal difference-based reinforcement learning algorithm that learns a set of Pareto non-dominated policies in a single run. Internally, it relies on a specific bootstrapping process that separates the immediate reward from the set of future rewards. This way, each vectorial estimate in the set can be updated as new information becomes available. The algorithm also comprises a mechanism to retrieve the state and action sequences that correspond to each of these policies.

Finally, we demonstrate the behaviour of a multi-policy algorithm on a simulation environment of a transmission system. For a transmission to be optimal, it needs to be both smooth and fast, which are in essence conflicting objectives. We analyse how the solution method explores the objective space and continually refines its knowledge.

Samenvatting

Reinforcement learning voorziet een raamwerk voor algoritmen die leren uit interacties met een onbekende omgeving. Sinds vele jaren hebben onderzoekers bijgedragen tot dit raamwerk door dergelijke algoritmen voor te stellen die hun gedrag optimaliseren over de tijd heen. Dit leerproces wordt gekarakteriseerd door het trachten maximaliseren van een enkel, scalair beloningsignaal. Echter zijn vele problemen uit de reële wereld van nature uit te complex om beschreven te worden door een enkel, scalair criterium. Doorgaans hebben deze problemen betrekking op meerdere doelstellingen die vaak aanleiding geven tot strijdige belangen. Een voorbeeld hiervan is te vinden in de industrie waar het doel van een onderneming is om te komen tot een beleid dat op hetzelfde moment de winst en de tevredenheid van de gebruiker maximaliseert, terwijl het eveneens de productiekosten, de arbeidskosten en de milieu-impact tracht te minimaliseren. Voor dit soort problemen is het traditionele reinforcement learning framework ontoereikend en ontbreekt het expressiviteit.

In dit proefschrift pleiten we voor de behoefte aan specifieke reinforcement learning algoritmen die speciaal zijn aangepast voor problemen met meerdere objectieven. Deze algoritmen heten vervolgens multi-objectieve reinforcement learning (MORL) algoritmen die een of meerdere Pareto optimale evenwichten van de doelstellingen verstrekken. Wat dit evenwicht intrinsiek uitstraalt is afhankelijk van de klemtoon en de voorkeuren van de beleidsmaker. In het geval dat de voorkeuren van de beleidsmaker duidelijk en bij voorbaat bekend zijn, kunnen single-policy technieken zoals scalarisatie functies gebruikt worden om het zoeken naar een bepaald compromis te begeleiden. In dit opzicht analyseren we verschillende instantiaties van scalarisatie functies zoals de lineaire en Chebyshev scalarisatie functie en we laten zien welke invloed ze hebben op de geconvergeerde oplossing.

In het geval dat de voorkeuren van de beleidsmaker onduidelijk zijn voor het optimalisatieproces plaatsvindt, zou het aangewezen zijn om een verzameling van Pareto optimale oplossingen aan de beleidsmaker voor te stellen die elk een ander evenwicht van de doelstellingen afweegt. Een mogelijkheid om een verzameling oplossingen te verkrijgen bestaat eruit de uitkomsten van een aantal scalarisatie functies, die elk uiteenlopende delen van

het zoekgebied onderzoeken, te verzamelen. In dit proefschrift stellen we twee algoritmen voor die dit principe volgen voor zowel discrete als continue Pareto ruimtes.

Een meer geavanceerd idee dat we ook nastreven bestaat eruit om een verzameling van compromissen te leren op een gelijktijdige basis. Om tegemoet te komen aan een daadwerkelijk multi-policy algoritme, breiden we de beginselen van het reinforcement learning kader uit, met inbegrip van de kennis representatie en het bootstrapping proces. Voor continue actie ruimtes, stellen wij het multi-objectief hiërarchische optimistisch optimalisatie algoritme voor dat stapsgewijs een binaire boom van vectoriële optimistische schattingen construeert. Voor sequentiële problemen introduceren we Pareto Q-learning. Pareto Q-learning is het eerste temporal difference-gebaseerd reinforcement learning algoritme dat een verzameling van Pareto dominerende oplossingen simultaan leert. Intern berust het algoritme op een specifiek bootstrapping proces dat de onmiddellijke beloning van de verzameling van de toekomstige beloningen scheidt. Op deze manier kan elke vectoriële schatting in de verzameling worden bijgewerkt als er nieuwe informatie beschikbaar wordt. Het algoritme omvat tevens een mechanisme om de opeenvolging van toestanden en acties die corresponderen met elk van deze geleerde oplossingen te achterhalen. Tot slot demonstreren we het gedrag van een multi-policy algoritme op een simulatieomgeving van een transmissiesysteem. Opdat een transmissie optimaal is, moet deze zowel soepel als snel zijn, welke in wezen strijdige doelstellingen zijn. We analyseren hoe deze oplossingsmethode de zoekruimte van het transmissiesysteem verkent en voortdurend zijn kennis verruimt.

Contents

Acknowledgements

This dissertation would not have been possible without the support and the encouragement of many people.

First and foremost I would like to express my special appreciation and thanks to my promoter Ann Nowé for giving me the opportunity to pursue a Ph.D. at the Computational Modelling lab. Thank you, Ann, for the scientific and personal guidance you have been giving me the past four years. I appreciate all the meetings, brainstorms and discussions that we had to continuously push the boundaries of my research.

Furthermore, I would also like to thank the members of the examination committee Karl Tuyls, Bernard Manderick, Kurt Driessens, Bart Jansen, Yann-Michaël De Hauwere and Coen De Roover for taking the time to read the dissertation and for your helpful insights and suggestions.

I am also grateful to the people I worked with on my two projects. For the Perpetual project, I would like to thank our colleagues at KULeuven en FMTC for our combined efforts on the industrial applications of multi-objective research. For the Scanergy project, I would like to personally thank Mike for hosting me three months in the offices of Sensing & Control in Barcelona. Also, thank you Narcís, Sam, Mihhail, Pablo, Raoul, José María, Sergio, Adrián, Helena, Pedro, Alberto and Angel for making it a memorable stay abroad!

I would also like to praise my colleagues at the CoMo lab. It has been a pleasure to spend four years in such good company. I will cherish the amazing moments that we had, such as our BBQs, the team building events in Ostend and Westouter, the after-hours drinks at the Kultuurkaffee and simply our easy chats in the lab. Thanks a lot Ann, Bernard, Anna, Abdel, Dip, Elias, Felipe, Frederik, Ivan, Ivomar, Isel, Jelena, Jonatan, Lara, Luis, Madalina, Marjon, Mike, Peter, Pieter, Roxana, Saba, Steven, The Ahn, Timo, Tom, Yailen, Yann-Michaël and Yami!

I deliberately left out three people from the list because they deserve some special attention. Maarten, Kevin and Tim, we started this long journey back in 2011 and encountered

so many beautiful, tough and inspiring moments. I could not have wished for better colleagues that became friends with a capital F (although Kevin will never admit it). I will never forget the (social dinners at the) BNAIC conferences, the AAMAS conference in Valencia, the football matches, the paintball that never took place, the personalised coffee mug, Boubakar, the quiz in the bar in Ostend, the break-in into Kevin's room, the daily ping-pong matches, the estimated 16000cl of free soup, the trips to Singapore, Miami, Japan, South-Africa and so much more!

Last but definitely not least, I would like to acknowledge the role of my family during my studies. I would like to express my gratitude to my parents and my brother for their never-ending and unconditional support. Their care allowed me to decompress after work and to forget the struggles I was facing. Also my friends had a great influence on me to unwind after a busy week. Thank you Wouter & Nele, Sander & Tine, Thomas, Maarten, Max and Ken! Also thank you, E.R.C. Hoeilaart and W.T.C. Hoeilaart for the football matches and the cycling races that allowed me to recharge my batteries during the weekend.

The last part of these acknowledgements is dedicated to the most important person in my life for more than 6 years. Lise, without you I was not able to find the devotion that was necessary for this kind of job. Due to your encouragement, I was able to overcome the most stressful moments. We shared both the high points and the low points of this four-year quest and I could not think of anyone else I wanted to share these moments with. Together with you, I will now focus on different objectives in life.

Brussels, March 2016.

1 Introduction

This dissertation is concerned with the study of reinforcement learning in situations where the agent is faced with a task involving multiple objectives. Reinforcement learning is a branch of machine learning in which an agent has to learn the optimal behaviour in an unknown environment. In essence, the agent has to learn over time what is the optimal action to take in each state of the environment. In reinforcement learning, the agent employs a principle of trial and error to examine the consequences actions have in particular states. It does so by analysing a scalar feedback signal associated to each action. Based on this signal, reinforcement learning tries to discover the optimal action for each particular situation that can occur.

However, many real-world problems are inherently too complex to be described by a single, scalar feedback signal. Usually, genuine decision making problems require the simultaneous optimisation of multiple criteria or objectives. These objectives can be correlated or independent, but they usually are *conflicting*, i.e., an increase in the performance of one objective implies a decrease in the performance of another objective and vice versa. An example can be found in the area of politics where parliamentarians have to decide on the location where to build a new highway that at the same time increases the efficiency of the traffic network and minimises the associated environmental and social costs. Other examples might not be limited to only two objectives. For instance, in the manufacturing industry, the goal of a company comprises many criteria such as maximising the profit and the customer satisfaction while minimising the production costs, the labour costs and the environmental impact. In these situations, traditional reinforcement learning approaches often fail because they oversimplify the problem at hand which on its turn results in unrealistic and suboptimal decisions.

In literature, different approaches exist that strive towards the development of specific multi-objective reinforcement learning techniques. However, most of these approaches are limited in the sense that they cannot discover every optimal compromise solution, leaving

potentially fruitful areas of the objective space untouched. Also, some of these approaches are not generalisable as they are restricted to specific application domains.

In this dissertation, we evaluate different approaches that allow the discovery of one or more trade-off solutions that balance the objectives in the environment. We group the algorithms in four categories. We first differentiate between single-policy and multi-policy techniques, based on the fact whether the algorithm learns a single compromise or a set of solutions, respectively. Furthermore, we also distinguish between the type of scalarisation function used in the process, i.e., being a linear scalarisation function or a monotonically increasing variant. We analyse the conceptual details of the algorithms and emphasize the associated design decisions. Additionally, we verify the theoretical guarantees and the empirical performance of these algorithms on a wide set of benchmark problems.

In the following sections, we will start off with the basics of artificial intelligence. We will define what agents are and what characterises an intelligent agent. Subsequently, we outline the problem statement and the contributions of this dissertation.

1.1 Agents

Nowadays, people often admit computer systems are brilliant and clever devices since they allow them to send and receive e-mail messages, organise agendas, create text documents and so much more. However, to offer this functionality, all the computer system does is processing some predefined lines of code that were implemented by a software engineer. If the computer system is asked to perform an unforeseen request that was not envisaged by the software engineer, the computer system will most likely not act as inquired or not act at all. Therefore, it is naive to think of these types of systems as being clever since they possess no reasoning capabilities and are not able to work autonomously. Yet, computer systems that do hold these properties are called *agents*.

In artificial intelligence (AI), the principle of an agent is a widely used concept. However, how an agent is defined differs throughout the different branches of AI. In general, the definition by Jennings et al. (1998) is considered a good summary of what comprises an agent:

> **Definition 1.1**
>
> An agent is a computer system that is situated in some environment, and that is capable of autonomous action taking in this environment in order to meet its design objectives.

Although this definition still remains very broad, the principle components of an agent structure are clear, i.e., the agent observes the environment it is operating in and selects an action, which in its turn can change the state of the environment, as illustrated in Figure 1.1.

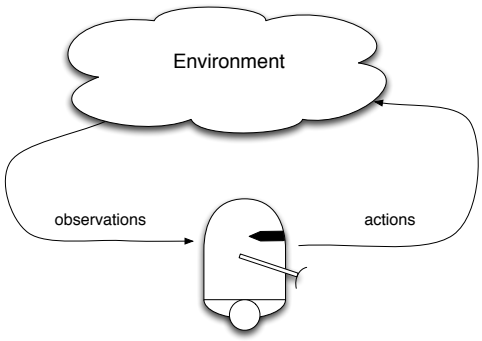


Figure 1.1: The agent acts upon its observations of the environment.

Other definitions make the features that constitute an agent more specific. For instance, in Weiß (1999), an *intelligent* agent is characterised by three additional requirements:

- *Reactivity* : The ability to perceive the environment and to respond in a timely fashion to changes that occur in order to satisfy its design objectives.
- *Pro-activeness* : The ability to take initiative in order to satisfy its design objectives.
- *Social ability* : The ability to interact with other agents and possibly humans in order to satisfy its design objectives.

The notion of pro-activity relates to the ability of the agent to express a specific behaviour that strives to achieve a certain goal. In the case the environment is static and fully known to the agent, the problem of finding a plan or recipe might be straightforward. When the environment contains multiple agents or when it comprises dynamic structures, the agent should continuously react to these changes. Ideally, the agent should be pro-active and reactive at the same time, i.e., acting upon the changes in the environment while expressing goal-directed behaviour. Since the environment can comprise multiple agents that do not necessarily share the same goals, the agent also needs to possess social skills. These skills allow the agents to coordinate and to negotiate with one another in order to meet their objectives. In the following section, we will make this definition more explicit by considering agents that have the ability to learn.

1.2 Learning agents

For an agent to possess the properties of reactivity, pro-activeness and social ability, it needs a mechanism to reason about how to solve the problem at hand. Reinforcement learning is a theoretical machine learning framework that permits agents to learn from their own experience (Sutton and Barto, 1998). This experience is characterised by interactions between the agent and a stochastic stationary environment which take place at every discrete time step $t = 1, 2, \ldots$. Each time step, the agent perceives the current state

of the environment and selects an action $a(t)$. Upon the application of this action, the environment transitions the agent into a new state $s(t+1)$ and provides an associated immediate reward $r(t+1)$. Internally, the reinforcement learning agent adapts the estimate of the quality of the action $a(t)$ into the direction of the obtained feedback signal. This process is illustrated in Figure 1.2. The goal of the agent is to learn a policy, i.e., a mapping from situations to actions, in order to maximise the rewards obtained.

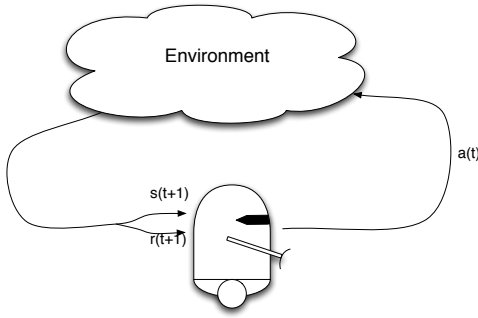


Figure 1.2: The reinforcement learning agent selects an action $a(t)$ which transitions the agent into a new state $s(t+1)$. At the same time the agent receives the scalar reward $r(t+1)$.

As time progresses and more and more actions are tried out, the estimates associated with each action become more and more precise. In the reinforcement learning framework, these estimates are guaranteed to converge to their expected value given certain assumptions, meaning that the agent can learn the optimal strategy over time that maximises the scalar feedback signal.

1.3 Problem statement

In this dissertation, we are concerned with problems that consist of multiple, conflicting objectives. In these problems, the feedback signal does not provide a scalar indication of the quality of an action but it is vectorial, i.e., one scalar reward for each objective. As an example, imagine the problem an investor is facing when trying to optimise his portfolio. At the same time, the investor is eager to maximise the possible returns of his investment while also maximising the security of the investment, i.e., minimising the risk. An illustration of the portfolio management problem can be found in Figure 1.3 where different types of investments are compared based on their performance on security and potential return, which are both to be maximised. A consequence of a vectorial evaluation of solutions is that there is no single optimal strategy but the agent assisting the investor can discover multiple trade-off policies that balance these objectives. In the illustration, we highlight these compromises as we see that some investments such as for instance saving

bonds (a) and precious metals (b) are very secure but offer only small profit margins. A counter example is the stock market (d) that has the potential of a high return of investment while at the same time a high probability in money losses as well. Real estate properties (c) offer a more balanced compromise between the two objectives. Since there are no solutions improving these four investments on any of the two objectives, these four approaches are said to be optimal trade-offs which the agent can discover. However, there are also solutions that are not optimal. Consider for instance, the investment in technological devices, such as computers (e), and cars (f), which rapidly devalue and take off in valuation.

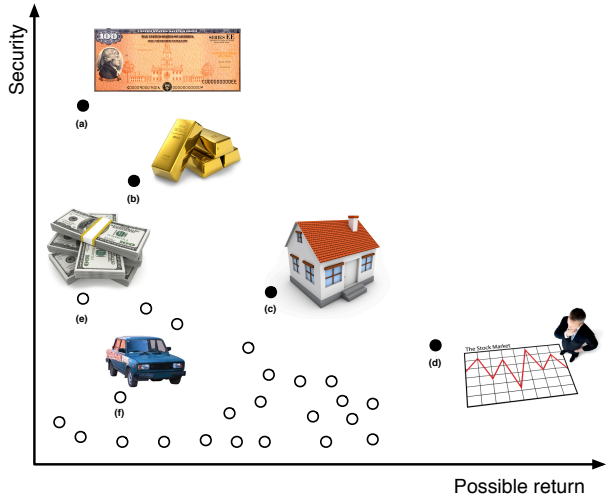


Figure 1.3: The trade-off between security and the potential return in a portfolio management problem. The optimal solutions (a) to (d) are denoted by black dots while the dominated solutions, such as (e) and (f) are white dots. The goal of the agent optimising the investment problem would be to discover the optimal compromise solutions (a) to (d).

Internally, the agent needs a mechanism to evaluate these multi-objective solutions based on the preferences of the investor. In some cases, these preferences can be very clear and the possible solutions can be totally ordered to select the best solution. For instance, the preference could be to maximize the possible return, regardless of the security objective. Hence, the agent would only suggest solution (d), i.e., the stock market. In other cases, when the preferences are unclear (or possibly even unknown), only a partial order can be retained. The mechanism that evaluates solutions while taking into account this preference information is called a scalarisation function. Based on the specific instantiation of the scalarisation function, different properties arise. Each of these properties will be covered in Chapter 3.

In general, multi-objective reinforcement learning approaches that learn these compromise solutions can be characterised based on two aspects. In Section 3.4, we provide a detailed

taxonomy, but for now it is sufficient to characterise the approaches based on (1) whether one or more solutions are returned and (2) whether these solutions are situated in a convex or non-convex region of the objective space.

A first and simple category of approaches learns a single, convex solution at a time. Since the solution is located in a convex region, mathematical formulas guarantee that the solution can be retrieved by a linear, weighted combination of the objectives. The effectiveness of this category is represented in Figure 1.4 (a) where both objectives need to be maximised. The blue dots represent dominated solutions while the set of optimal solutions, referred to as the Pareto front, is depicted in black. The areas in red denote a subset of the Pareto front that can be discovered by convex combinations. In this red area, the white dot represents a possible solution the techniques in this category would learn.

A second category represents a collection of techniques that learn a set of multiple, convex solutions. These mechanisms exploit the same mathematical properties of the algorithms of the first category, but this time to retrieve a set of trade-off solutions. Each of these solutions then lies in convex areas of the Pareto front, as depicted in Figure 1.4 (b). Ideally, this category of algorithms would aim to discover the entire set of solutions covering the red areas.

Although a significant amount of research in reinforcement learning has been focussing on solving problems from the viewpoint of these situations (Vamplew et al., 2010; Roijers et al., 2013), only a subset of the Pareto front is retrievable, leaving potential fruitful parts of the objective space untouched. A large part of this dissertation addresses the problem of discovering solutions that cannot be defined by convex combinations. In that sense, we analyse the theoretical and the empirical performance of reinforcement learning algorithms for retrieving optimal solutions, that lie in both convex and non-convex parts of the Pareto front. This constitutes the third and fourth category. The third category encompasses algorithms that learn a single convex or non-convex solution at a time, as depicted in Figure 1.4 (c).

The fourth category involves algorithms for learning multiple solutions simultaneously. These algorithms then retrieve a set of both convex and non-convex solutions as presented in Figure 1.4 (d). In essence, these algorithm are genuine multi-policy algorithms that can discover the entire Pareto front of optimal solutions, as they are not limited to a specific subset.

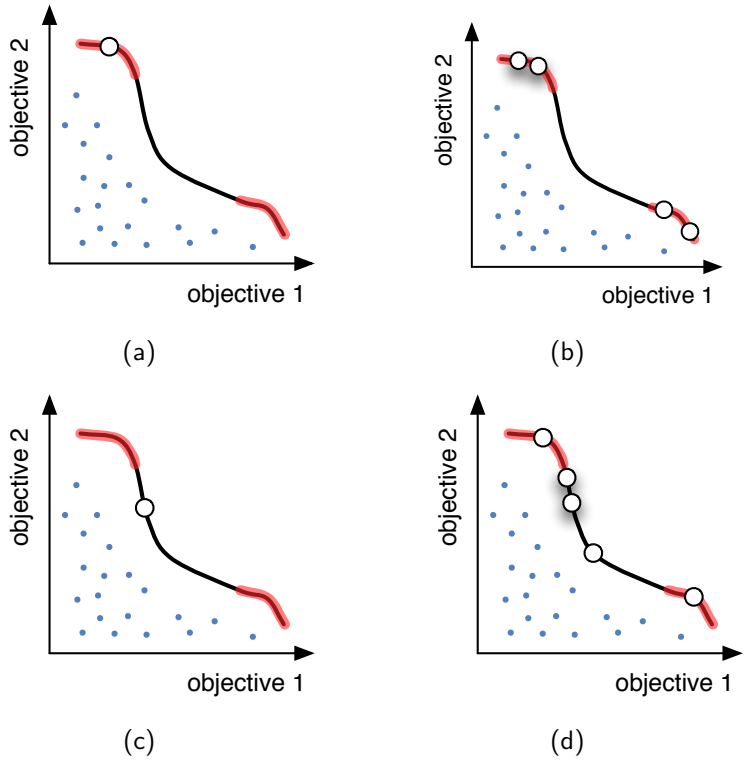


Figure 1.4: Research in multi-objective reinforcement learning is categorised in four clusters based on the type and the amount of solutions learned.

As such, we can formulate the research question that motivates the work in this thesis:

> **Research question**
>
> How can agents solve multi-objective reinforcement learning problems and learn one or more Pareto optimal trade-off solutions that balance several criteria?

Throughout this thesis, we provide three responses to this research question, based on the context of the multi-objective problem. Firstly, in Chapter 4, we assume a context where only a single solution is retrieved at a time, i.e., situations (a) and (c) in Figure 1.4. We research how reinforcement learning can be extended to learn solutions in convex and non-convex regions of the Pareto front and what the limitations of these techniques are. Secondly, in Chapter 5, we propose two algorithms that iteratively learn multiple solutions based on linear combinations, similar to the context of Figure 1.4 (b). Finally, in Chapter 6, we present two algorithms that learn multiple optimal solutions at a time that

are located in both convex and non-convex regions of the Pareto front. These approaches relate to Figure 1.4 (d).

1.4 Contributions

While answering the research question, we have made several contributions to the field of multi-objective reinforcement learning. We summarise the main achievements in the following list:

- We provide an overview of the foundations and the history of standard, single-objective reinforcement learning. We closely examine the theoretical characteristics of the reinforcement learning framework and explicitly describe all the components that compose a reinforcement learning problem.
- We provide a taxonomy of multi-objective reinforcement learning research based on the type of scalarisation function used in the decision making process and the amount of solutions learned.
- We highlight the advantages and limitations of current state-of-the-art approaches within this field. We argue that there is a need for algorithms that go beyond the exclusive search for convex solutions. We show that these solutions only represent a subset of the Pareto front, leaving other optimal trade-off solutions undiscovered.
- We propose a framework for single-policy algorithms that rely on scalarisation functions to reduce the dimensionality of the multi-objective feedback signal to a scalar measure. In this framework, the knowledge representation of the agent is extended to store a vector of estimates and the scalarisation function is employed in the action selection process.
- In that same framework, we argue that the choice of the scalarisation function is crucial since it has a huge impact on the solution the reinforcement learning technique converges to. In this respect, we theoretically and empirically analyse a linear and non-linear scalarisation function on their effectiveness in retrieving optimal trade-off solutions.
- We analyse how the weight parameters of the linear scalarisation function relate to specific parts of the objective space. We highlight that the mapping from weight space to objective space is non-isomorphic, meaning that it is far from trivial to define an appropriate weight configuration to retrieve a desired trade-off solution.
- In this regard, we propose two adaptive weight algorithms (AWA) that iteratively adjust their weight configurations based on the solutions obtained in previous rounds. These algorithms have the ability to adaptively explore both discrete and continuous Pareto fronts.
- We surpass single-policy scalarisation functions by investigating algorithms that simultaneously learn a set of Pareto optimal solutions. We first propose MO-HOO, an algorithm that is tailored for single-state environments with a continuous action space. Internally, MO-HOO constructs and refines a tree of multi-objective estimates that cover the actions space. For multi-state environments we propose Pareto

Q-learning, which is the first temporal difference based multi-policy algorithms that does not employ a linear scalarisation function. The main novelty of Pareto Q-learning is the set-based bootstrapping rule which decomposes the estimates into two separate components.

- We go beyond academic benchmark environments and analyse the multi-policy MO-HOO algorithm on a simulation environment of a wet clutch, a transmission unit that can be found in many automotive vehicles. In this setup, the goal is to find an engagement that is both fast and smooth, which are conflicting criteria.

1.5 Outline of the dissertation

In this section, we provide an outline of the content of each of the chapters of this dissertation:

In Chapter 2, we introduce the foundations of single-objective reinforcement learning. We present its history and foundations, which lie in the fields of psychology and optimal control theory. Additionally, we define the components that comprise a reinforcement learning setting such as value functions and policies. We also describe the notion of a Markov decision process, which is a mathematical framework to model the environment the agent is operating in. Furthermore, we emphasise the synergies and the difference between model-based and model-free learning for sequential decision making problems. Since the remainder of this dissertation is focussed on model-free learning, we pay particular attention to the Q-learning algorithm, which is one of the foundations of the studies in the following chapters.

In Chapter 3, we present the idea of learning in environments that require the optimising multiple criteria or objectives. We introduce the general concepts of multi-criteria decision making and we provide a survey of the theoretical foundations of multi-objective optimisation. With special care, we describe the characteristics of the Pareto front and the articulation of the preferences of the decision maker. In this regard, we categorise multi-objective reinforcement learning algorithms based on a taxonomy that classifies these techniques based on the type of scalarisation function, i.e., a linear or a more general monotonically increasing scalarisation function, and the amount of solutions learned, i.e., single-policy or multi-policy. We situate the current state-of-the-art reinforcement learning algorithms within this problem taxonomy, together with our contributions within this dissertation.

In Chapter 4, we elaborate on multi-objective reinforcement learning algorithms that employ a scalarisation for learning a single policy at a time. We first present a general framework for single-policy reinforcement learning, based on the Q-learning algorithm. In this framework, we analyse the established linear scalarisation function and we draw attention to its advantages and limitations. Additionally, we elaborate on an alternative scalarisation function that possesses non-linear characteristics. This is the Chebyshev scalarisation function, which is well-known in the domain of multi-objective optimisation. We

stress the lack of convergence guarantees of the Chebyshev function within the framework on reinforcement learning on a theoretical level. Furthermore, we investigate to what degree this limitation holds in practice on several benchmark instances.

In Chapter 5, we explore the relation between the weight configuration, which incorporates predefined emphasis of the decision maker, and the learned policy in the objective space. Based on this mapping, we investigate how single-policy algorithms can be augmented to obtain multiple solutions by iteratively adapting this weight configuration. We propose two algorithms that rely on tree structures to produce a series of weight configurations to obtain a set of optimal solutions. The first algorithm is specifically tailored for Pareto fronts with a discrete amount of solutions while the second algorithm is capable of retrieving solutions with a high degree of diversity in continuous Pareto fronts.

In Chapter 6, we present two distinct algorithms for learning various compromise solutions in a single run. The novelty of these algorithms lies in the fact that they can retrieve all solutions that satisfy the Pareto relation, without being restricted to a convex subset. A first algorithm is the multi-objective hierarchical heuristic optimisation (MO-HOO) algorithm for multi-objective $\mathcal{X}$-armed bandit problems. This is a class of problems where an agent is faced with a continuous finite-dimensional action space instead of a discrete one. MO-HOO is a single-state learning algorithm that is specifically tailored for multi-objective industrial optimisation problems that involve the assignment of continuous variables. A second algorithm is Pareto Q-learning (PQL). PQL extends the principle of MO-HOO towards multi-state problems and combines it with Q-learning. PQL is the first temporal difference-based multi-policy MORL algorithm that does not use the linear scalarisation function. Pareto Q-learning is not limited to the set of convex solutions, but it can learn the entire Pareto front, if enough exploration is allowed. The algorithm learns the Pareto front by bootstrapping Pareto non-dominated $\mathbf{Q}$-vectors throughout the state and action space.

In Chapter 7, we go beyond the academic benchmark environments and apply a collection of our algorithms on a real-life simulation environment. This environment simulates the filling phase of a wet clutch which are used in shifting and transmission systems of automotive vehicles. We analyse the application of a multi-policy reinforcement learning algorithm. More precisely, we apply the multi-objective hierarchical optimistic optimisation (MO-HOO) algorithm on the simulation environment in order to approximate its Pareto front.

In Chapter 8, we conclude the work presented in this dissertation and we outline avenues for future research.

A graphical outline of the dissertation is provided in Figure 1.5. The arrows indicate the recommended reading order.

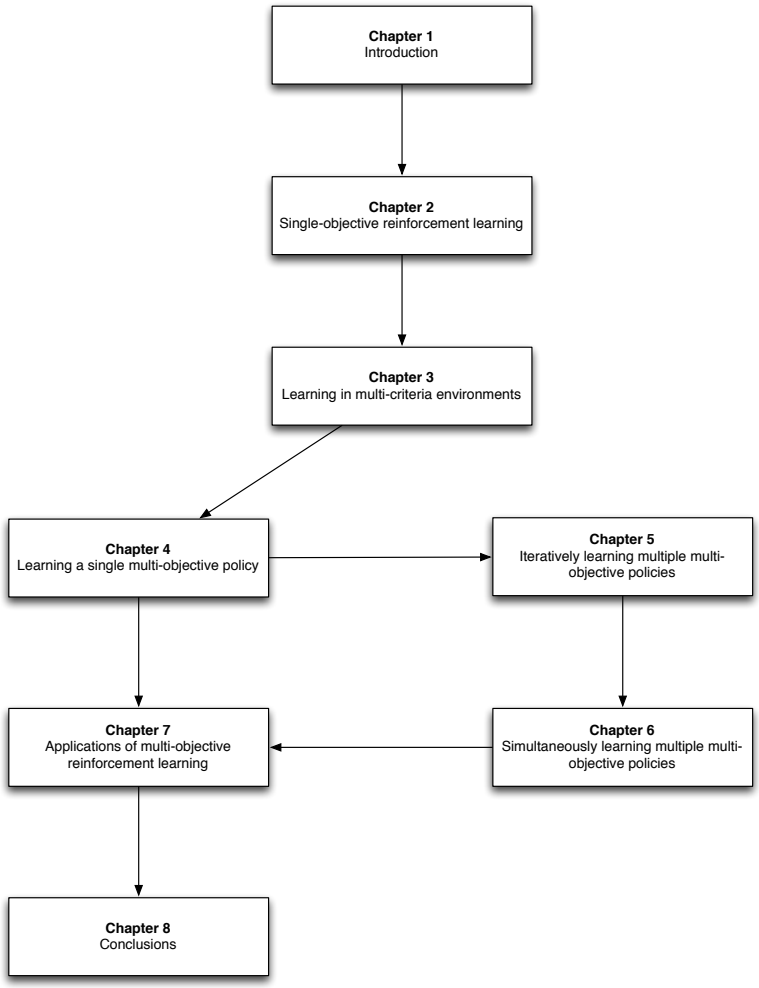


Figure 1.5: Graphical outline of the dissertation.

2 | Single-objective reinforcement learning

In reinforcement learning (RL), the agent is faced with the problem of taking decision in unknown, possibly dynamic environments. In the standard single-objective case, the overall goal of the agent is to learn the optimal decisions so as to maximise a single, numerical reward signal. These decisions are in essence actions that have to be selected in certain states of the environment. This environment is usually defined using a *Markov Decision Process*. This is a mathematical framework used to model the context of the problem the agent is solving. We describe this framework in Section 2.2. As the environment itself may be evolving over time, the consequence of a particular action on the resulting behaviour might be highly unpredictable. Also, the effect of the actions might not be direct but *delayed*. Therefore, the agent has to learn from its experience, using *trial-and-error*, which actions are fruitful.

In this chapter we introduce some preliminary concepts on single-objective reinforcement learning that we use as a basis throughout this dissertation. Firstly, we elaborate on the historical background of reinforcement learning which lies in the field of *animal learning* and *optimal control*. We also explain the theoretical foundations of reinforcement learning and explicitly define all the components that compose a reinforcement learning problem. We will focus on both *stateless* as well as *stateful* problems. Subsequently, this general introduction will be augmented with an experimental analysis of several action selection strategies in a classical reinforcement learning problem. In the end, we conclude the chapter by summarising the most important aspects.

2.1 History of reinforcement learning

Reinforcement learning is learning from interaction with an unknown environment in order to take the optimal action, given the situation the agent is facing (Sutton and Barto, 1998). The agent is not supported by a teacher pointing out the optimal behaviour or correcting sub-optimal actions, but it is solely relying on itself and its past experience in the environment. This experience is quantified by the agent's previously selected actions in the states of the environment and the observed reward signal. This principle of learning dates back to two main research areas which evolved rather independently from each other. These two tracks are *animal learning* and *optimal control*.

Psychologists observing animals have already found out over a hundred years ago that these creatures exhibit learning behaviour. A first experiment was conducted by Edward L. Thorndike, where he placed a cat in a wooden box and let it try to escape (Thorndike, 1911). In the beginning, the cat tried various approaches until it accidentally hit the magical lever that allowed it to flee. When Thorndike later put the cat again in this cage, it would execute the correct action more quickly than before. The more Thorndike repeated this experiment, the faster the cat escaped the box. In essence, the cat executed a type of *trial-and-error* learning where it would learn from its mistakes and its successes (Schultz and Schultz, 2011). These mistakes and successes are then quantified by a feedback signal, i.e., in the case of the cat, this signal would reflect if the action led to the cat's freedom or not. Sometimes the effect of an action can be direct, like in the case of the cat, but in some situations the outcome can also be *delayed*. For instance, in the game of chess, the true influence of a single move may not be clear at the beginning of the game, as only at the end of the game a reinforcement is given for winning or losing the game. Hence, the delayed reward needs to be propagated in order to favour or disfavour the course of actions in case of winning or losing the game, respectively. Another distinctive challenge that characterises reinforcement learning is the so-called *exploration-exploitation trade-off*. This trade-off comprises one of the major dilemmas of the learning agent: when has it acquired enough knowledge in the environment (exploration), before it can use this information and select the best action so far (exploitation), given that the environment is stochastic and perhaps even changing over time? It is clear that if the agent explores too much, it might unnecessarily re-evaluate actions that it has experienced numerous times to result in poor performance. When the agent explores too little, it is not guaranteed that the action the agent *estimates* to be the best policy is also the best policy in the environment. In Section 2.4.2, we will elaborate on some established techniques for dealing with this issue.

The second research track that is a foundation of reinforcement learning is the discipline of optimal control. In optimal control, the goal is to find a sequence of actions so as to optimise the behaviour of a dynamic system over a certain time period. One of the principal discoveries within this field was the Bellman equation (Bellman, 1957). The Bellman equation defines a complex problem in a recursive form and serves as a foundation of many *Dynamic Programming* (DP) algorithms (Bertsekas, 2001a,b). Dynamic programming is a mechanism for solving a complex problem by breaking it apart into many subproblems

and will be covered in more detail in Section 2.4.1. First, we describe a mathematical framework that serves as a model for sequential decision making. This framework is a Markov decision process.

2.2 Markov decision processes

In sequential decision making, Markov decision processes (MDP) are used to define the environment the agent is operating in. MDPs provide a mathematical model to describe a discrete-time state-transition system (Puterman, 1994). At each time step t, the agent is in a certain state $s(t)$ and must select an action $a(t)$. Upon selecting this action, the environment transitions the agent into a new state $s(t+1)$ and the agent receives a corresponding reward $r(t+1)$. The transition from state $s(t)$ to state $s(t+1)$, given action $a(t)$, is based on a probability distribution defined as the transition function T. Formally, a Markov decision process is described as follows:

Definition 2.1

A **Markov decision process** is a 4-tuple (S, A, T, R), where

- $S = s^1, \ldots, s^N$ is a finite set of states,
- $A = \cup_{s \in S} A(s)$ where $A(s)$ is a finite set of available actions in state $s \in S$,
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function $T(s'|s, a)$ specifying the probability of arriving in state s' after selecting action a in state s,
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function $R(s, a, s')$ specifying the expected reward associated to transitioning from state s with action a to state s'.

It is important to note that this transition to $s(t+1)$ only depends on the previous state $s(t)$ and not on any previously visited states. This property is called the Markov Property:

Definition 2.2

A system is said to possess the **Markov Property** if the future state transitions of the system are independent of the previous states, given the current state:

$$T(s(t+1)|s(t, a(t), \ldots, s(0), a(0)) = T(s(t+1)|s(t), a(t))$$

In general, we distinguish between two different types of MDPs. These are *infinite horizon* and *finite-horizon* MDPs. In finite-horizon MDPs, the state space S consists of *terminal* states, i.e., states that end the problem, and no more reward can be obtained. Hence, the problem is said to be *episodic* as it consists of several episodes where the agent initiates its decision making loop from a start state to a terminal state. Usually, video game environments are defined by finite-horizon problems: the episode starts with the character at a particular location in the world and the episode ends if it reaches a checkpoint.

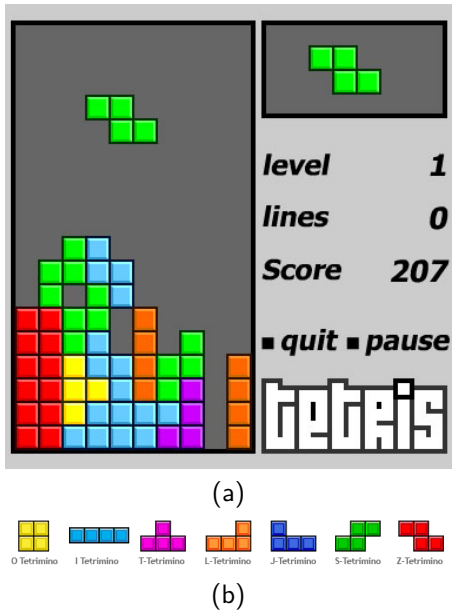
In some situations, the exact length of the agent's lifetime might not be known in advance. These problems are referred to as infinite-horizon problems since they have got a *continuous* nature. An example of such a problem is the cart pole problem (Sutton and Barto, 1998). In this problem, a robot tries to balance a pole on a cart by applying force to that cart. Based on the angle of the pole and the velocity of the cart, the robot has to learn when to move left or right along a track. Clearly, for this problem it is not possible to state beforehand where or when the agent's lifespan comes to a halt.

The overall goal of the agent is to learn a policy, a mapping from states to actions. We are not just interested in any policy, but in an optimal policy, i.e., a policy that maximises the expected reward it receives over time. A formal definition of a policy will be provided in Section 2.3 where we present general notations of reinforcement learning. We conclude the definition of MDPs by presenting an illustrative example.

Example 2.1

In this example, we consider the Tetris game, a puzzle-based game released the 1980s by Alexey Pajitnov that is still very popular today . In Tetris, geometric shapes, called *Tetriminos*, need to be organised in the matrix-like play field (see Figure 2.1 (a)), called the *wall*. There are seven distinct shapes that can be moved sideways and rotated 90, 180 and 270 degrees. A visual representation of the different Tetriminos can be found in Figure 2.1 (b). These shapes need to be placed in order to form a horizontal line without any gaps. Whenever such a line is created, the line disappears and blocks on top of the horizontal line fall one level down. Many versions of the Tetris game exist, including different levels and different game speeds, but the goal of the Tetris game in this example is to play as long as possible, i.e., until a piece exceeds the top of the play field. This game can be modelled in terms of an Markov decision process as follows:

- The states in the MDP represent the different configurations of the play field and the current Tetriminos being moved. The configuration can be described by the height of the wall, the number of holes in the wall, difference of height between adjacent columns and many more (Gabillon et al., 2013; Furmston and Barber, 2012).
- The actions in the MDP are the possible orientations of the piece and the possible locations that it can be placed on the play field.
- The transition function of the MDP define the next wall configuration deterministically given the current state and action. The transitions to the next Tetrimino are given by a certain distribution, e.g., uniformly over all possible types.
- The reward function of the MDP provides zero at all times, except when the height of the wall exceeds the height of the play field. In that case, a negative reward of -1 returned.



(a)

(b)

Figure 2.1: The Tetris game (a) and its seven geometrical shapes called Tetriminos (b).

2.3 Notations of reinforcement learning

In the previous section, we have presented a definition of the environment the agent is operating in. The next part of the puzzle is to define the goal of the agent in that environment, that is, to act in such a way in order to maximise the reward it receives over time. In this section, we formally describe this goal in mathematical terms. Subsequently, we also specify how the agent gathers knowledge on the fruitfulness of its action and how this knowledge evolves over time. We present two distinct types of learning, i.e., *model-based* and *model-free* learning, depending on whether a model of the environment is available to the agent or not. In this dissertation, we concentrate on model-free learning. Therefore, in Section 2.4.2, we present some additional techniques on how to gather knowledge in an incremental manner and how to tackle the exploration-exploitation problem, discussed earlier in Section 2.1.

In reinforcement learning problems, the agent is in a certain state of the environment $s(t)$ and has to learn to select the action $a(t)$ at every time step $t = 0, 1, \ldots$. In return, the environment transitions the agent into a new state $s(t+1)$ and provides a numerical reward $r(t+1)$ representing how beneficial that action was. It is important to note that the transition to $s(t+1)$ and the observed reward $r(t+1)$ are not necessary (and also not likely) to be deterministic. On the contrary, these are usually random, stationary variables. This means they are random with respect to a certain distribution, but the distribution is not changing over time.

Definition 2.3

In a **stationary** environment, the probabilities of making state transitions or receiving reward signals do not change over time.

A schematic overview of the RL agent is given in Figure. 2.2.

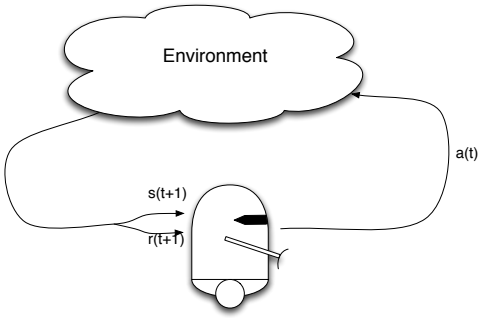


Figure 2.2: The reinforcement learning model, based on (Kaelbling et al., 1996).

The goal of this agent is to act in such a way that its expected *return*, R_t, is maximised. In the case of a finite-horizon problem, this is just the sum of the observed rewards. When the problem consists of an infinite-horizon, the return is an infinite sum. In this case, a *discount* factor, $\gamma \in [0, 1[$, is used to quantify the importance of short-term versus long-term rewards. Consequently, the discount factor ensures this infinite sum yields a finite number:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.1}$$

When γ is close to 0, the agent is *myopic* and mainly considers immediate rewards. Instead, if γ approaches 1, the future rewards are more important. For finite-horizon MDPs, the sum might be undiscounted and γ can be 1.

In order to retrieve this return, the agent needs to learn a *policy* $\pi \in \Pi$, where Π is the space of policies. A policy is a function that maps a state to a distribution over the actions in that state. Several types of policies exist, depending on the type of distribution and on the information it uses to make its decision. Here we distinguish between two properties, i.e., (1) *deterministic* or *stochastic* and (2) *stationary* or *non-stationary*. Each of these types of policies will be defined in the following paragraph.

A deterministic policy is a policy that deterministically selects actions.

Definition 2.4

A **deterministic** policy $\pi \in \Pi$ is a policy for which the probability of selecting an action a in state s at time step t, i.e., $\pi(s, a)$, equals 1 while for every other action $a' \in A$, the probability is 0

$$\exists a \in A(s) : \pi(s, a) = 1 \wedge \forall a' \neq a \in A : \pi(s, a') = 0$$

A stochastic policy is a policy that selects actions according to a probability distribution (Nowé et al., 2012). Thereby, deterministic policies are a subset of stochastic policies.

Definition 2.5

A **stochastic** policy $\pi \in \Pi$ is a policy for which the probability of selecting an action a in state s at time step t, i.e., $\pi(s, a)$ is larger or equal to 0 while ensuring that the sum of probabilities equals 1.

$$\forall a \in A(s) : \pi(s, a) \geq 0 \wedge \sum_{a \in A(s)} \pi(s, a) = 1$$

A stationary policy is a policy that selects actions based on the current state only (Puterman, 1994).

Definition 2.6

A **stationary** policy $\pi \in \Pi$ is a function for which the input is only the current state s, i.e.,

$$\pi : S \to A.$$

A non-stationary policy is a policy that does not select actions based on the current state only, but also on the time-step t. Hence, the selection of an action does not only depend on the state the agent is in, but also on the absolute time the agent has been operating in the environment (Kaelbling et al., 1995). For instance, in a robot environment, the action could be selected depending on lifespan of the agent. Generally speaking, it is to be expected that the agent would select different actions at the last day of its lifespan than at the beginning.

Definition 2.7

A **non-stationary** policy $\pi \in \Pi$ is a function for which the input is the current state s and the a time step t, i.e.,

$$\pi : S \times t \to A.$$

In standard, single-objective reinforcement learning (SORL), an optimal policy $\pi^* \in \Pi$ of the underlying MDP is a deterministic stationary policy (Bertsekas, 1987). In the upcoming chapters of this dissertation we will see that this is not the case when the reward signal is composed of multiple objectives.

Based on the stationary principle in SORL, the agent can learn a policy that is *globally* optimal, i.e., over multiple states, by taking actions *locally* for each state individually. The notion of the *value function* is important for this matter.

Definition 2.8

The **value function** $V^\pi(s)$ specifies how good a certain state s is in the long term according to the policy π. The function returns the expected, discounted return to be observed when the agent would start in state s and follow policy π:

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s(t) = s\} \\ &= E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s(t) = s \right\} \end{aligned}$$

The value of an action a in a state s under policy $\pi \in \Pi$ is represented by a $Q^\pi(s, a)$. This is called a Q-*value*, referring to the *quality* of an action in a state and can be defined as follows.

Definition 2.9

The **value of an action** a **in a state** s, $Q^\pi(s, a)$ is the expected return starting from s, taking action a, and thereafter following policy π:

$$\begin{aligned} Q^\pi(s, a) &= E_\pi\{R_t | s(t) = s, a(t) = a\} \\ &= E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s(t) = s, a(t) = a \right\} \end{aligned}$$

In the previous sections, we have introduced a theoretical framework that forms the basis of reinforcement learning. We have formally defined the environment the agent is operating in and how the agent represents its knowledge. In the following sections we will describe how we can learn the optimal Q-values, Q^*, which will be used to learn the optimal policy, π^*.

2.4 Learning methods

Two main categories of learning approaches exist. These are *model-based* and *model-free* techniques. In model-based learning, an explicit model of the environment is used to compute the optimal policy π^*. This model can either be available *a priori* to the agent to exploit or it can be learned over time. In the former case, *dynamic programming* is

the advised solution technique, while in the latter case *dyna architecture* methods, which concise an integrated architecture for learning, planning, and reacting, can be used (Sutton, 1991). Model-based approaches will be discussed in Section 2.4.1.

In model-free learning, no explicit model of the environment is considered, but the agent learns the optimal policy in an implicit manner from interaction. As we will see in Section 2.4.2, model-free learning uses the same mathematical foundations as model-based learning.

2.4.1 Model-based learning

In a model representation of the environment the knowledge on the transition function $T(s'|s,a)$ and the reward function $R(s,a,s')$ are explicit. When these aspects of the environment are known to the agent, dynamic programming methods can be used to compute the optimal policy π^*. Dynamic programming will be explained in the subsequent sections.

Dynamic programming

Richard Bellman introduced dynamic programming as a process that solves a complex problem by breaking it down into subproblems. This definition relates closely to the process of solving an MDP by computing a policy which consists of local action selections in every state.

An important discovery within this field is the *Bellman equation*. The Bellman equation (Bellman, 1957) is a recursive function used to calculate the value of a state s according to a certain policy $\pi \in \Pi$, i.e., $V^\pi(s)$:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_a T(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]. \tag{2.2}$$

This recursive equation gives the utility of a state in terms of the average immediate reward and a weighted average of the utility of the successor states based on the transition function.

Since the value function of a policy provides a scalar value, policies can be ranked in a total order. This order relation is defined as follows:

Definition 2.10

A policy $\pi \in \Pi$ is **better than or equal to** a policy $\pi' \in \Pi$ if its expected return is **greater than or equal to** the expected return of π' over all states, i.e.,

$$\pi \geq \pi' \text{ iff } V^\pi(s) \geq V^{\pi'}(s), \forall s \in S.$$

There is always a policy $\pi^* \in \Pi$ that is the *optimal* policy of an MDP. The value function of this policy is greater than or equal to the value function of all policies over all states.

$$V^*(s) = \max_\pi V^\pi(s), \forall s \in S \tag{2.3}$$

Similar to the definition of the optimal value function of a state, we can also define the *optimal state-action* value function Q^* over all states and all actions:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a), \forall s \in S, a \in A(s). \tag{2.4}$$

This equation gives us the expected return for selecting an action a in state s and thereafter following the optimal policy π^*. Hence,

$$Q^*(s,a) = E_\pi\{r(t+1) + \gamma V^*(s(t+1)) | s(t) = s, a(t) = a\}. \tag{2.5}$$

Since $V^*(s)$ is the value of a state under the optimal policy, it is equal to the expected return for the best action of that state. This is referred to as the *Bellman optimality equation* of a state s and is defined as follows.

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\ &= \max_{a \in A(s)} E_{\pi^*}\{R_t | s(t) = s, a(t) = a\} \\ &= \max_{a \in A(s)} E_{\pi^*}\left\{ \sum_{k=0}^{\infty} \gamma^k r(t+k+1) | s(t) = s, a(t) = a \right\} \\ &= \max_{a \in A(s)} E_{\pi^*}\left\{ r(t+1) + \gamma \sum_{k=1}^{\infty} \gamma^k r(t+k+1) | s(t) = s, a(t) = a \right\} \\ &= \max_{a \in A(s)} E\{r(t+1) + \gamma V^*(s(t+1)) | s(t) = s, a(t) = a\} \\ &= \max_{a \in A(s)} \sum_{s'} T(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \end{aligned} \tag{2.6}$$

The Bellman optimality equation for state-action pairs, Q^*, is defined in a similar way:

$$\begin{aligned} Q^*(s,a) &= E\{r(t+1) + \gamma \max_{a'} Q^*(s(t+1), a') | s(t) = s, a(t) = a\} \\ &= \sum_{s'} T(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q^*(s', a')]. \end{aligned} \tag{2.7}$$

Algorithms that solve this equation to calculate the optimal policy are called dynamic programming algorithms. Over the years many dynamic programming algorithms have been proposed, but the two main approaches are *policy iteration* and *value iteration*. Policy iteration uses an alternation of evaluation and improvement steps to calculate the value of the entire policy. Value iteration does not need a separate evaluation step but is able to compute the value of a policy iteratively using a simple backup scheme. For more information on these two principle dynamic programming algorithms, we refer to (Bellman, 1957; Puterman, 1994; Bertsekas, 2001a,b).

Note: It is important to note that the Bellman equation is based on the property of *additivity*, meaning that passing the sum of two variables through a function f is equal to summing the result of f applied to the two variables individually.

Definition 2.11

A function f is additive if it preserves the addition operator:

$$f(x + y) = f(x) + f(y)$$

This is a basic, but an important requirement as we will see once we equip multi-objective learning algorithms with scalarisation functions in Chapter 4.

2.4.2 Model-free learning

In the previous section we have examined techniques to compute the optimal policy when a model of the environment is available. In this section, we explore the model-free approach, i.e., the agent does not have at its dispense a model of the environment, but it has to learn what to do in an *implicit* manner through interaction. This is referred to as learning from experience. In this section, we will present a well-known algorithm, called Q-learning, that does exactly this.

Q-learning

One of the most important advances in reinforcement learning was the development of the Q-learning algorithm (Watkins, 1989; Watkins and Dayan, 1992). In Q-learning, a table consisting of state-action pairs is stored. Each entry contains a value for $\hat{Q}(s, a)$ which is the learner's current estimate about the actual value of $Q^*(s, a)$. In the update rule, the Q-value of a state-action pair is updated using its previous value, the reward and the values of future state-action pairs. Hence, it updates a guess of one state-action pair with the guess of another state-action pair. This process is called *bootstrapping*. The $\hat{Q}$-values are updated according to the following update rule:

$$\hat{Q}(s, a) \leftarrow (1 - \alpha(t))\hat{Q}(s, a) + \alpha(t)[r + \gamma \max_{a'} \hat{Q}(s', a')], \tag{2.8}$$

where $\alpha(t) \in [0, 1]$ is the learning rate at time step t and r is the reward received for performing action a in state s. The pseudo-code of the algorithm is listed in Algorithm 1.

In each episode, actions are selected based on a particular action selection strategy, for example ϵ-greedy where a random action is selected with a probability of ϵ, while the greedy action is selected with a probability of $(1 - \epsilon)$. Upon applying the action, the environment transitions to a new state s' and the agent receives the corresponding reward r (line 6). At line 7, the $\hat{Q}$-value of the previous state-action pair (s, a) is updated towards the reward r and the maximum $\hat{Q}$-value of the next state s'. This process is repeated until the $\hat{Q}$-values converge or after a predefined number of episodes.

To conclude, Q-learning and general model-free learning possess a number of advantages:

- It can learn in a fully incremental fashion, i.e., it learns whenever new rewards are being observed.

Algorithm 1 Single-objective Q-learning algorithm

Initialise $\hat{Q}(s, a)$ arbitrarily
for each episode t **do**
    Initialise s
    repeat
        Choose a from s using a policy derived from the $\hat{Q}$-values (e.g., ϵ-greedy)
        Take action a and observe $s' \in S$, $r \in \mathbb{R}$
        $\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha_t(r + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a))$
        $s \leftarrow s'$
    until s is terminal
end for

- The agent can learn *before* and *without* knowing the final outcome. This means that the learning requires less memory and that it can learn from incomplete sequences, respectively, which is helpful in applications that have very long episodes.
- The learning algorithm has been proven to converge under reasonable assumptions. Provided that all state-action pairs are visited infinitely often and a suitable evolution for the learning rate is chosen, the estimates, $\hat{Q}$, will converge to the optimal values, Q^* (Tsitsiklis, 1994).

At first sight, visiting all state-action pairs infinitely often seems hard and infeasible in many applications. However, the amount of exploration is usually reduced once the Q-values have converged, i.e., when the difference between two updates of the same Q-value does not exceed a small threshold. In the next section, we will discuss some well-known techniques for balancing exploration and exploitation in model-free environments.

Action selection strategies

As we have already discussed in Section 2.1, the exploration-exploitation trade-off is one of the crucial aspects of reinforcement learning algorithms. This dilemma is the issue of deciding when the agent has acquired enough knowledge in the environment (exploration), before it can use this information and select the best action so far (exploitation). Below, we elaborate on the four main techniques that leverage a certain trade-off of exploration and exploitation.

- *Random action selection* : This is the most simple action selection strategy. In this case, the agent does not take into account its $\hat{Q}$-estimates, but it selects an action a randomly from the set $A(s)$. It is obvious that this action selection strategy is not a mechanism to let the agent obtain high-quality rewards (exploitation), but it is a simple technique to let the agent explore as it selects uniformly between the actions in a state.
- *Greedy action selection* : In this strategy, the agent selects the action that it believes is currently the best action based on its estimate $\hat{Q}$. However, if the agent follows

this strategy too early in the learning phase, this estimate might be inaccurate and the resulting policy might be sub-optimal.

- ϵ-*greedy action selection* : This mechanism is a simple alternative to acting greedy all the time. In an ϵ-greedy action selection, the agent selects a random action with a probability of ϵ and a greedy action with a probability of $(1 - \epsilon)$. It is important to note that the ratio of exploration and exploitation is static throughout the entire learning phase.
- *Softmax action selection* : In order to leverage this ratio of exploration and exploitation, a probability distribution can be calculated for every action at each action selection step, called a *play*. The softmax action selection strategy selects actions using the Boltzmann distribution. The probability of selecting an action a in state s is defined at follows:

$$P(a) = \frac{e^{\frac{\hat{Q}(s,a)}{\tau}}}{\sum_{a' \in A(s)} e^{\frac{\hat{Q}(s,a')}{\tau}}}, \tag{2.9}$$

where $\tau \in \mathbb{R}_0^+$ is called the temperature. In the case of high values for τ, the probability distribution of all actions are (nearly) uniform. When a low value is assigned to τ, the greedier the action selection will be. Usually, this parameter is changed over time where in the beginning of the learning phase a high value for τ will be chosen which is being degraded over time.

2.5 Reinforcement learning in stateless environments

After we explained the theoretical foundations of reinforcement learning, it might be appropriate to evaluate these techniques in a practical context. Therefore, we turn ourselves to a classical problem in reinforcement learning, called the *multi-armed bandit* problem. In the multi-armed bandit problem, or also called the *n-armed bandit* problem, a gambler is faced with a collection of slot machines, i.e., arms, and he has to decide which lever to pull at each time step (Robbins, 952). Upon playing a particular arm, a reward is drawn from that action's specific distribution and the goal of the gambler is to maximise his long-term scalar reward.

This problem is a *stateless* problem, meaning that there is no state in the environment.[1] Therefore, we do not need an implementation of the entire Q-learning algorithm of Section 2.4.2, but we can use a simpler updating rule. We can update the estimate $\hat{Q}$ using the sample average method. The $\hat{Q}(a)$-value, representing the estimate of action a is given by:

$$\hat{Q}(a) = \frac{r(0) + r(1) + \ldots + r(k)}{k}, \tag{2.10}$$

where $r(i)$ is the reward received at time step i and k is the number of times action a has been selected as yet. Initially, the $\hat{Q}$-value of each action is assigned 0. Once the agent performs an action selection, called a play, the episode is finished.

[1] or only a single state, depending on the viewpoint

Although this problem seems trivial, it is not. The multi-armed bandit problem provides a clear mapping to the exploration-exploitation dilemma of Section 2.1; the agent has to try to acquire new knowledge while at the same time use this knowledge to attempt to optimise the performance of its decisions. Extensive research has been conducted for this problem in the field of packet routing (Awerbuch and Kleinberg, 2008), ads placement (Pandey et al., 2007) and investment and innovation (Hege and Bergemann, 2005).

2.5.1 Experimental evaluation

In this section, we experimentally compare the action selection strategies of Section 2.4.2 on the multi-armed bandit problem. More precisely, we analyse the following strategies:

- Random action selection
- Greedy action selection
- ϵ-greedy, $\epsilon = 0.1$
- ϵ-greedy, $\epsilon = 0.2$
- Softmax, $\tau = 1$
- Softmax, $\tau = 0.1$
- Softmax, $\tau = 1000 * 0.9^{play}$
- Softmax, $\tau = 1000 * 0.95^{play}$

We will analyse their behaviour on a 4-armed bandit problem. The expected reward of each arm is given below in Table 2.1. The rewards are drawn from a normal distribution with mean Q_a^* and the variance equal to 1. Furthermore, the initial Q-values are assigned to zero and we averaged the performance of the action selection strategies over 1000 trials of each 1000 plays.

Action	Q_a^*
action #1	1.7
action #2	2.1
action #3	1.5
action #4	1.3

Table 2.1: The Q_a^* for each action of the multi-armed bandit problem.

We first depict the behaviour of the action selection strategies in terms of three criteria: (1) the probability over time of selecting the optimal action, which is in this case action #2, (2) the average reward received over time and (2) the number of times each action is selected. These results are presented in Figures 2.3, 2.4 and 2.5, respectively. We summarise the results for each of the traditional action selection strategies below.[2]

[2]For n-armed bandits also more specialised action selection strategies exist that analyse the upper confidence bound of each arm (Auer et al., 2003; Audibert et al., 2009)

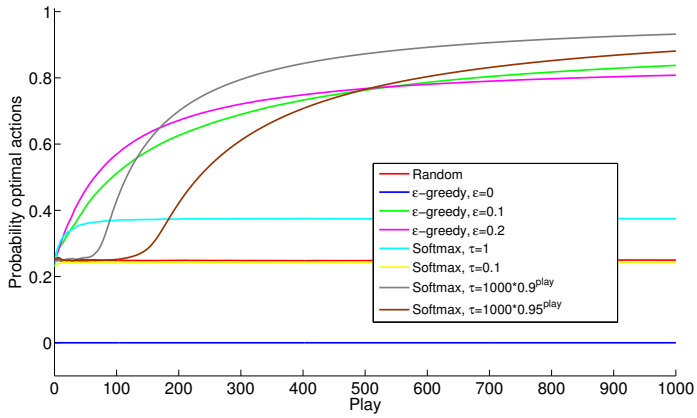


Figure 2.3: The probability of selecting optimal actions for each of the action selection strategies in the multi-armed bandit problem.

- *Random action selection* : As this strategy does not consider the obtained knowledge in the $\hat{Q}$-values and selects actions uniformly at random, the probability of selecting the optimal action is $\frac{1}{4}$ as there are 4 actions in this environment (see Figure 2.3 and Figure 2.5). Because the strategy is very naive and not exploiting, the average reward obtained over time in Figure 2.4 is also very low.
- *Greedy action selection* : Selecting the greedy action all of the time is also not an ideal strategy. In Figure 2.3, we see that the greedy action is always stuck and never finds the best action. As, we see in Figure 2.5, action $\#1$ is selected all of the times, because the algorithm never explores other levers than the one it selected the first time. Hence, only for this action the $\hat{Q}$-value approaches the Q^*-value, while for the other actions, the $\hat{Q}$-value remains its initial value.
- *ϵ-greedy action selection* : Introducing the ϵ-parameter clearly allows us up to some extent to balance exploration and exploitation. In Figure 2.3, we see that introducing an ϵ parameter of 0.1 allows us in the end to reach a performance of 84%. Utilising a higher probability for random actions with $\epsilon = 0.2$ degrades the performance in terms of the optimal actions and the average reward compared to $\epsilon = 0.1$.
- *Softmax action selection* : We see that the algorithms with $\tau = 0.1$ and $\tau = 1$ do not perform well. With a high τ value, the algorithm is exploring a lot while a low τ value makes the algorithm behave greedily and it gets stuck in a local optimum. Therefore, it is recommended to let the temperature parameter degrade over time. This idea is implemented when assigning $\tau = 1000 * 0.9^{play}$ and $\tau = 1000 * 0.95^{play}$. From the figures, we see that the strategies explore a lot in early learning phases while exploiting more at the end. For this particular case, we see that when assigning $\tau = 1000 * 0.9^{play}$ yields the best performance in terms of selecting the optimal action and the obtained reward.

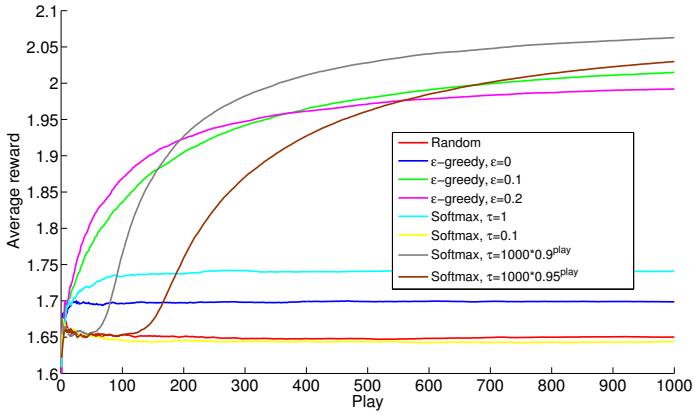


Figure 2.4: The average reward obtained for each of the action selection strategies in the multi-armed bandit problem.

The total accumulated reward obtained for each of the action selection strategies is presented in Table 2.2. This statistic also shows that the softmax strategy found the best compromise between explorative and exploitative behaviour over the entire learning period. The ϵ-greedy strategies approach its performance due to their rapid increase in early learning phases, but this does not suffice to surpass the performance of the softmax algorithm.

Strategy	**Cumulative reward**
Random	1649.8
Greedy	1698.3
ϵ-greedy, $\epsilon = 0.1$	1950.1
ϵ-greedy, $\epsilon = 0.2$	1947.6
Softmax, $\tau = 1$	1737.6
Softmax, $\tau = 0.1$	1644.4
Softmax, $\tau = 1000 * 0.9^{play}$	**1976.5**
Softmax, $\tau = 1000 * 0.95^{play}$	1903.1

Table 2.2: The total accumulated reward for each of the action selection strategies in the multi-armed bandit problem.

Even in this simple, stateless example, we have seen that it is crucial to find a good balance between exploration and exploitation. Behaving too greedily and exploiting too early tends to lead to suboptimal performance at the end of the learning task, while unnecessary exploration does not allow to obtain a lot of reward during learning. Although the softmax algorithm provided the best performance for this learning task, this result

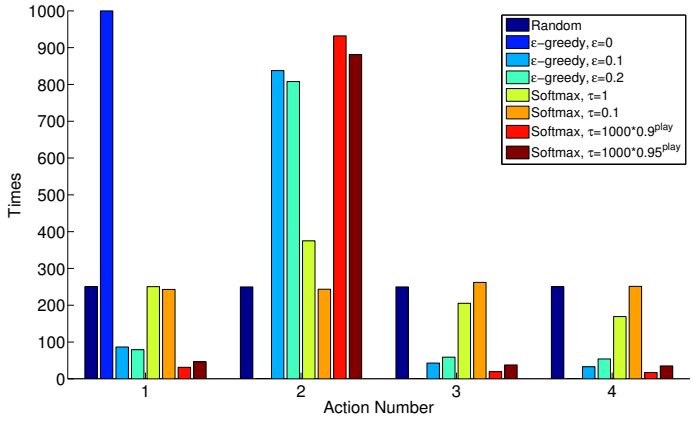


Figure 2.5: The number of times each action is selected for each action selection strategy.

cannot be generalised. Up to this day, there is no general answer to solving the exploration-exploitation dilemma as it requires tuning that is problem-specific (Auer et al., 2002). In the following chapter, we will analyse the additional difficulties that arise when the environment is no longer single-objective, but when multiple reward signals are supplied to the agent.

2.6 Summary

In this chapter, we introduced the foundations of single-objective reinforcement learning. We presented its history, which lies in the fields of psychology and optimal control theory. We explained the crucial aspects that define a reinforcement learning problem and described all its components, such as the Markov decision process and definitions of a policy and a value function. We paid special attention to the Bellman equation which is the foundation of many reinforcement learning methods. We also emphasised the synergies and the differences between model-based and model-free learning for sequential decision making problems, i.e., situations where the model is either known or unknown to the agent. For model-free learning approaches, the Q-learning algorithm was one of the major breakthroughs in the late 1980s.

Furthermore, we introduced different action selection strategies that allow to balance the exploration towards uncharted territory and the exploitation of the acquired knowledge. These mechanisms were tested on a simple, single-state problem, called the n-armed bandit problem.

3 | Learning in Multi-Criteria Environments

Many real-world problems require the simultaneous optimisation of multiple criteria or objectives. These objectives can be either correlated, conflicting or independent. In this chapter we present several characteristics for learning in these environments. We introduce the general concepts of *multi-criteria decision making* and we provide a survey of the theoretical foundations of multi-objective optimisation. With special care, we introduce *evolutionary multi-objective* (EMO) algorithms that use population-based principles to offer a wide range of advantages in multi-objective environments. Many of these EMO algorithms have also inspired research to solving multi-objective problems with reinforcement learning techniques. This research area is called *multi-objective reinforcement learning* (MORL) and is the topic of this dissertation. We formally specify the differences between single-objective reinforcement learning and MORL in Section 3.3 and highlight why specific MORL algorithms are required for solving this particular set of problems. Finally, we present an overview of MORL research that is categorised by two aspects and we highlight our main contributions in Section 3.4.

3.1 Introduction

Many real-world decision making problems are inherently too complex to be described by a single aspect or criterion. Restricting ourselves to these one-dimensional approaches would be a simplification of the problem at hand which would result in unrealistic and suboptimal decisions. Usually, these decision making problems deal with the systematic and simultaneous optimisation of a collection of objectives (Coello Coello et al., 2006; Tesauro et al., 2008; Hernandez-del Olmo et al., 2012). These objectives can be correlated

or independent, but usually they are *conflicting*, i.e., an increase in the performance of one objective implies a decrease in the performance of another objective and vice versa. For example, in the area of portfolio management, the manager wants to maximise the returns while at the same time reduce the risks. It is clear that a risky policy has the potential of a high return of investment while at the same time a high probability in money losses as well (Hassan, 2010). In vehicle routing, the goal is to generate a tour on a network that minimises the overall distance of the route, the workload of the driver, the customer satisfaction and many more (Jozefowiez et al., 2008). Additionally, in a wireless sensor network the packets need to be routed in such a way that both energy consumption and latency is optimised (Gorce et al., 2010).

In such cases, there is no single optimal solution, but the idea is to come up with *trade-off* or *compromise* solutions that balance one or more objectives. As such, we are dealing with a *multi-criteria decision making* (MCDM) problem. MCDM is a process that analyses complex problems that possess multiple, possible conflicting objectives (Zionts, 1979). In the following section, we will accurately define the fundamental components of MCDM.

3.1.1 Definitions of Multi-Criteria Decision Making

In this section, we will formally introduce multi-criteria decision making (MCDM) and multi-objective optimisation, which define the key foundations of problem solving in multiple dimensions. Belton and Stewart define MCDM as follows (Belton and Stewart, 2002):

> **Definition 3.1**
>
> MCDM is an umbrella term to describe a collection of formal approaches which seek to take explicit account of multiple criteria in helping individuals or groups explore decisions that matter.

In other words, MCDM is concerned with the development of methodologies to solve complex decision making problems involving multiple goals of conflicting nature. Although MCDM problems have been typical and relevant for all problems of mankind, research in the area was relatively minimal until the last 40 years. Up to today, MCDM is a vivid research track that has generated significant amount of methods and techniques for multi-criteria problems (Stewart, 1992; Figueira et al., 2005; Tzeng and Huang, 2011; Aruldoss et al., 2013).

3.1.2 Multi-Objective Optimisation

Multi-objective optimisation (MOO) is a subarea of MCDM where each objective can be defined by means of a mathematical equation and constraints. Without loss of generality it can be assumed that these objectives are to be maximised. A general multi-objective

optimisation problem can be formulated by an *objective function* $\mathcal{F}$:

$$\max \mathcal{F}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})), \tag{3.1}$$

where m is the number of objectives and $\mathbf{x} = (x_1, x_2, \ldots, x_k)$ is the *decision vector* in the decision space $\mathbf{X}$. The decision vector $\mathbf{x}$ then defines the input to the optimisation problem. The objective function f_i evaluates the decision vector on objective i and assigns it a real value, i.e., $f_i : \mathbf{X} \to \mathbb{R}$. Thus, the *objective function* $\mathcal{F}$ provides an *objective vector* $(y_1, y_2, \ldots, y_m)$, where each index i contains the value of $f_i(\mathbf{x})$. Furthermore, constraints might be specified on the objective functions that define a minimal threshold for the objective function value f_i for objective i. In order to find a *feasible* solution to the problem, this constraint should be met. The constraint function $g : \mathbf{X} \to \mathbb{R}$, where $g(\mathbf{x}) \leq 0$ is to be satisfied. Usually, multi-objective optimisation problems - and also the works presented in this dissertation - deal with problems where $m \in \{2, 3\}$. In the case of $m > 3$, we speak of many-objective optimisation. This is a separate research track aimed at solving problems with very high dimensionality that require specific multi-criteria solution methods (Deb and Jain, 2014; Jain and Deb, 2014).

Defining optimal solutions

So far, we have formalised the components that define a MOO problem. The goal of solving such a problem is to identify a decision vector $\mathbf{x} \in \mathbf{X}$ such that $\forall \mathbf{x}' \in \mathbf{X}, \mathcal{F}(\mathbf{x}') \prec \mathcal{F}(\mathbf{x})$. Therefore, it is crucial to define a binary relation $\prec$ on the objective space. In a single-objective problem, defining such a relation is easy; the solutions are evaluated by a single scalar value which entails that we can totally order the solutions as in Figure 3.1 (a). However, in the multi-objective case, we are no longer optimising in a single-dimension but the objective function provides a vectorial evaluation of a solution. Therefore, typically, there is no single global solution in MOO problems, but there exist a set of solutions that meet a certain optimality criterion (see Figure 3.1 (b)). The traditional relation that compares two solutions is the *Pareto dominance* relation (Pareto, 1896).

A solution (strictly) Pareto dominates another solution if it is strictly better on at least one objective of another solution x', while not being strictly worse on the other objectives:

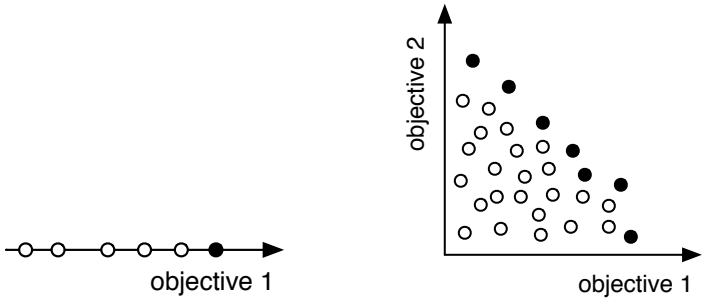
Definition 3.2

A solution $\mathbf{x} \in \mathbf{X}$ **strictly Pareto dominates** another solution $\mathbf{x}' \in \mathbf{X}$, i.e., $\mathbf{x}' \prec \mathbf{x}$ when

$$\mathbf{x}' \prec \mathbf{x} \iff \exists j : f_j(\mathbf{x}) > f_j(\mathbf{x}') \wedge \forall i \neq j : f_i(\mathbf{x}) \not< f_j(\mathbf{x}'). \tag{3.2}$$

This binary relation possesses a few properties:

- Irreflexive: The Pareto dominance relation is not reflexive since it does not hold that a solution $\mathbf{x}$ Pareto dominates itself.



(a) (b)

Figure 3.1: In the single-objective case in (a), the solutions of the problem can be totally ordered. In the case of two or more dimensions, the solutions are vectorial and no longer totally ordered. The Pareto relation specifies a partial order. The optimal solutions are denoted by black dots in both figures.

- Asymmetric: The Pareto dominance relation is not symmetric since $\mathbf{x} \prec \mathbf{x}'$ does not imply $\mathbf{x}' \prec \mathbf{x}$.
- Antisymmetric: Since the Pareto dominance relation is not symmetric, it cannot be antisymmetric.
- Transitive: The Pareto dominance relation is transitive as when $\mathbf{x} \prec \mathbf{x}'$ and $\mathbf{x}' \prec \mathbf{x}''$, then $\mathbf{x} \prec \mathbf{x}''$ also holds.

The definition of strict Pareto dominance can be relaxed. A solution $\mathbf{x}$ weakly Pareto dominates another solution $\mathbf{x}'$, i.e., $\mathbf{x}' \preceq \mathbf{x}$ when there does not exist an objective where $\mathbf{x}'$ is better than $\mathbf{x}$.

Definition 3.3

A solution $\mathbf{x} \in \mathbf{X}$ **weakly Pareto dominates** another solution $\mathbf{x}' \in \mathbf{X}$, i.e., $\mathbf{x} \preceq \mathbf{x}'$ when

$$\mathbf{x}' \preceq \mathbf{x} \iff \forall j : f_j(\mathbf{x}) \geq f_j(\mathbf{x}'). \tag{3.3}$$

The Pareto relation defines a partial order amongst the solutions. Two solutions $\mathbf{x}$ and $\mathbf{x}'$ are *incomparable* if $\mathbf{x}$ does not Pareto dominate $\mathbf{x}'$ and vice versa:

Definition 3.4

Two solutions $\mathbf{x}, \mathbf{x}' \in \mathbf{X}$ are **incomparable**, i.e., $\mathbf{x} \parallel \mathbf{x}'$ when

$$\mathbf{x} \parallel \mathbf{x}' \iff \mathbf{x} \not\prec \mathbf{x}' \wedge \mathbf{x}' \not\prec \mathbf{x}. \tag{3.4}$$

A solution $\mathbf{x}$ is *non-dominated* by another solution $\mathbf{x}'$ if it is either Pareto dominating or incomparable with that solution:

Definition 3.5

A solution $\mathbf{x} \in \mathbf{X}$ is **non-dominated** by another solution $\mathbf{x}' \in \mathbf{X}$, i.e., $\mathbf{x} \not\prec \mathbf{x}'$ when

$$\mathbf{x} \not\prec \mathbf{x}' \iff \mathbf{x}' \prec \mathbf{x} \vee \mathbf{x}' \parallel \mathbf{x}. \tag{3.5}$$

The goal of solving MOO problems is to come up with solutions that are Pareto optimal, i.e. the solutions that are non-dominated by any other solution:

Definition 3.6

A solution $\mathbf{x} \in \mathbf{X}$ is **Pareto optimal** iff it is non-dominated by any other solution $\mathbf{x}' \in X$:

$$\mathbf{x} \text{ is Pareto optimal} \iff \forall \mathbf{x}' \in \mathbf{X} : \mathbf{x} \not\prec \mathbf{x}'. \tag{3.6}$$

The set of decision vectors that define Pareto optimal solutions is called the *Pareto optimal set*, while the set of objective vectors of these solutions is called the *Pareto front*. In Figure 3.1 (b), the white dots represent solutions that are dominated by the black dots, which define the Pareto front. The Pareto dominance is a commonly accepted optimality criterion for multi-objective problems (Bäck, 1996; Knowles and Corne, 2002; Vamplew et al., 2010). Nevertheless, also other optimality criteria exist, such as ϵ-dominance (Laumanns et al., 2002), ordering cones (Batista et al., 2011), etc. Especially in multi-objective reinforcement learning, these alternatives play a crucial role. For more information on optimality criteria, we refer to Section 3.1.2.

Characteristics of the Pareto front

The Pareto optimal set and the Pareto front are closely related and possess some interesting characteristics that can have an impact on the multi-criteria decision making process. In this section, we highlight the main properties of the Pareto front, which are the *range*, the *shape* and the *size*.

The range of the Pareto front represents the dimensions of the Pareto front. For instance, in a two-dimensional environment, the range can provide details on the width and the height of the area that holds the Pareto optimal solutions. This area can be represented by two points, i.e., the *ideal* point and the *nadir* point:

Definition 3.7

Let $Y \subseteq X$ be the set of Pareto optimal solutions of a maximisation problem. The **ideal point** $\mathbf{z} = (z_1, z_2, \ldots, z_m) \in \mathbb{R}^m$ of this problem is then defined as:

$$z_i = \max_{y \in Y} f_i(y). \tag{3.7}$$

Definition 3.8

Let $Y \subseteq X$ be the set of Pareto optimal solutions of a maximisation problem. The **nadir point** $\mathbf{z} = (z_1, z_2, \ldots, z_m) \in \mathbb{R}^m$ of this problem is then defined as:

$$z_i = \min_{y \in Y} f_i(y) \tag{3.8}$$

In Figure 3.2, we depict a graphical representation of both points. Based on these coordinates, the decision maker has more knowledge of the area and size of the objective space that contains the optimal solutions, which can be helpful for example in interactive optimisation processes, where the user can interactively steer the search process (Deb et al., 2006).

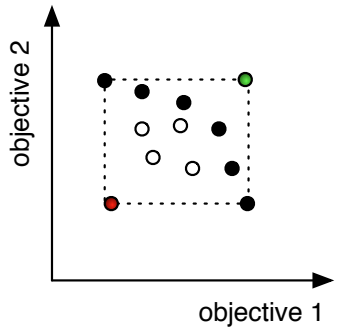
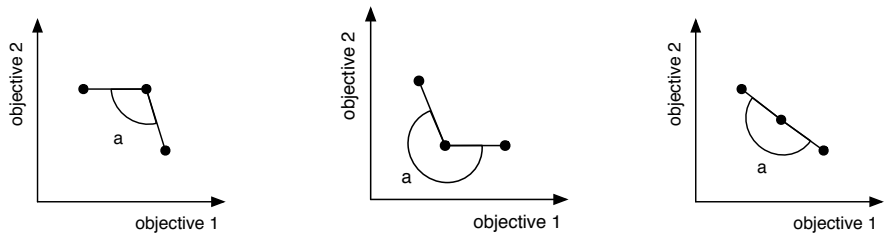


Figure 3.2: The Pareto front of a maximisation problem. The back dots denote the Pareto front that dominate the white dots. The red and green dots depict the nadir and ideal point, respectively.

The size of the Pareto front can also have an impact on the optimisation process. When the size of the Pareto front is relatively large, traditional techniques might have a difficult time approaching the entire set of optimal solutions. The number of objectives of a problem also directly relates to the size of the Pareto front. As proven in Winkler (1985), the number of Pareto incomparable solutions increases with the number of objectives.

The shape of the Pareto front is another important property that influences the optimisation process. The form of the Pareto front reflects how strongly the optimal solutions



(a) (b) (c)

Figure 3.3: When the internal angle a of all indentations in the Pareto front is either smaller, bigger or equal to 180 degrees, the shape of the Pareto front is convex (a), concave (b) or linear (c), respectively. Both objectives are to be maximised.

are correlated. We differentiate between a discrete and a continuous decision space. If the decision space is discrete, the set of Pareto solutions is not contiguous and contains a finite number of elements as in Figure 3.2. When the decision space is continuous, the shape of the Pareto front represents a line that connects all optimal solutions without any gaps (Ehrgott, 2005). Since the shape of the Pareto front also influences the effectiveness of particular MORL algorithms that rely on (linear) scalarisation functions, as we will see in Section 4.2.2, it is appropriate to review this property below. We distinguish between five principal types of Pareto fronts, depending on their shape:

- Convex shape : A Pareto front is convex in shape if the shape of the Pareto front is bulging outwards. Another way to determine the shape is to look at the internal angle that is defined by every three adjacent vectors as denoted in Figure 3.3 (a). When no internal angle is greater than 180 degrees, the Pareto front is convex (Yaochu, 2003). An example of a convex Pareto front can be found in Figure 3.4 (a). More formally, a Pareto front is convex if a normalised linear combination of two of its elements y_1 and y_2 is always lower than a normalised third element y_3:

  Definition 3.9

  Let $Y \subseteq X$ be the set of Pareto optimal solutions of a maximisation problem. The Pareto front is **convex** if :

  $$\forall y_1, y_2 \in Y \wedge \forall 0 \leq \lambda \leq 1, \exists y_3 \in Y : \lambda \|y_1\| + (1-\lambda)\|y_2\| \leq \|y_3\|. \quad (3.9)$$

- Concave shape: A Pareto front is concave in shape if the shape of the Pareto front is bulging inwards. Alternatively, it should also hold that there are indentations where the internal angle is greater than 180 degrees, as shown in Figure 3.3 (b). [1] An example of a concave Pareto front can be found in Figure 3.4 (b). Mathematically

[1] Concave or non-convex are synonyms.

speaking, a Pareto front is concave if a normalised linear combination of two of its elements y_1 and y_2 is always larger than a normalised third element y_3:

Definition 3.10

Let $Y \subseteq X$ be the set of Pareto optimal solutions of a maximisation problem. The Pareto front is **concave** if :

$$\forall y_1, y_2 \in Y \wedge \forall 0 \leq \lambda \leq 1, \exists y_3 \in Y : \lambda \|y_1\| + (1-\lambda)\|y_2\| \geq \|y_3\|. \quad (3.10)$$

- Linear shape: A Pareto front is linear if the line representing the Pareto front can be analytically defined by $m \cdot x + b$, where m is the slope of the line and b is the intercept. Logically, the internal angle is then equal to 180 degrees for every triple of vectors (Figure 3.3 (c)). An example of a linear Pareto front can be found in Figure 3.4 (c). Technically, a Pareto front is linear if a normalised linear combination of two of its elements y_1 and y_2 is equal to a normalised third element y_3:

Definition 3.11

Let $Y \subseteq X$ be the set of Pareto optimal solutions of a maximisation problem. The Pareto front is **linear** if :

$$\forall y_1, y_2 \in Y \wedge \forall 0 \leq \lambda \leq 1, \exists y_3 \in Y : \lambda \|y_1\| + (1-\lambda)\|y_2\| = \|y_3\|. \quad (3.11)$$

- Disconnected shape: In some problems, the Pareto front might not be continuous but discontinuous, i.e., there might exist gaps as in Figure 3.1 (d). Problems that retain a disconnected Pareto front are particularly difficult for multi-objective optimisation techniques as these methods no longer guarantee to converge to the global Pareto front if it contains gaps (Huband et al., 2006).

Pareto fronts can also contain combinations of the above shapes. For instance, in Figure 3.4 (e), we depict a Pareto front that contains a mixture of both convex and concave shapes.

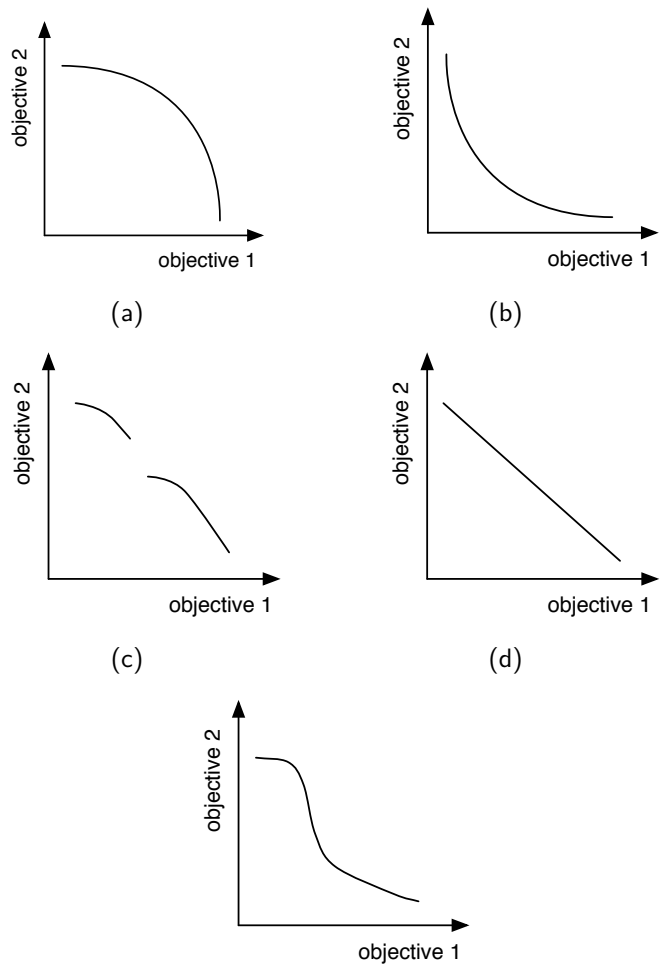


Figure 3.4: Several Pareto fronts with different shapes.

Other characteristics of the Pareto front are not listed here. The interested reader can find a more comprehensive discussion of other properties in Ehrgott and Gandibleux (2002); Figueira et al. (2005).

To conclude, we denote a practical MOO problem in an illustrative manner. This is the problem of acquiring a new vehicle.

Example 3.1

When a person is looking to buy a vehicle, he might be interested in a vehicle that is both cheap and speedy. In other words, the person would like to find a vehicle that maximises the speed objective and minimises the cost objective. In Figure 3.5, we depict 12 vehicles and their performance in terms of these two objectives. We note that in this example, there is no single vehicle that is both the fastest and the cheapest, but there are several trade-off solutions: vehicles that balance the speed and cost objective in a certain way. In our example, we clearly see that vehicle (f) is not an ideal solution as every other vehicle is better in terms of speed. This solution, together with the other solutions denoted by white dots are Pareto dominated by the black dots. These five vehicles are Pareto optimal and mutually incomparable as they provide an optimal trade-off. These vehicles then define the Pareto optimal set while their evaluations on the speed and cost level define the Pareto front.

Depending on the specific trade-off the person is interested in, e.g., preferring cost over speed or vice versa, the optimal decision would be to acquire one of these five vehicles.

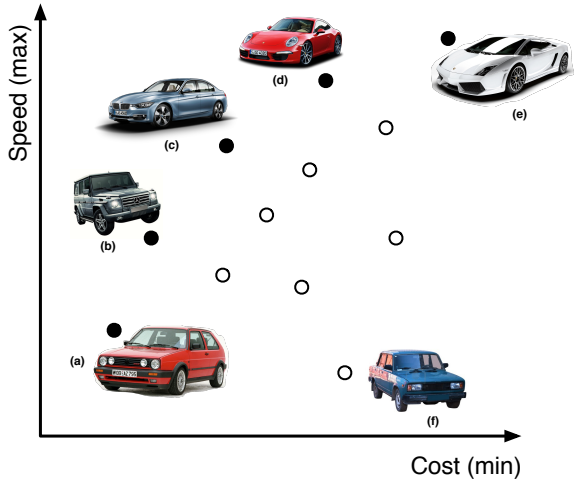


Figure 3.5: Vehicles can be evaluated on several criteria. The vehicles (a) to (e), denoted by a black dot, possess optimal combinations of speed and cost, while the white dots, such as vehicle (f), are not optimal, but dominated.

Articulation of preferences

So far, we have presented the principal theoretical concepts of multi-criteria optimisation, such as the Pareto optimal set and the Pareto front. We now introduce another major

part of the MCDM process, more precisely the role of the decision maker. The decision maker possesses preferences or opinions concerning the solutions in the objective space. These opinions reflect the relative importance of the different objectives, e.g, the preferred balance between speed and cost in Example 3.1. Whether deliberately or not, these preferences are applied in the mind of the decision maker to evaluate the results obtained by the optimisation process, in order to finally select a single solution in the execution phase. In literature, we discover three distinct ways how the preferences of the decision maker can be used in the optimisation process to select a single solution in the objective space. These are *a priori*, *a posteriori* and *interactive* articulation of preferences. We will highlight these types of preferences which will be used to introduce the algorithmic notions of *single-policy* and *multi-policy* categories.

In the case of a priori articulation of preferences, the decision maker provides its preferences before the optimisation process takes place. Many of these a priori methods incorporate preferences in the form of parameters that steer the search process to solutions that satisfy these preferences. Usually, a *scalarisation* function is employed that reduces the original multi-objective problem to a single-objective problem, taking into account the preferences of the decision maker as constraints to refine the optimality criterion. A scalarisation function f is a function that projects a vector to a scalar, given a certain weight vector:

Definition 3.12

A **scalarisation function** f is a function that projects a vector $\mathbf{v}$ to a scalar:

$$v_{\mathbf{w}} = f(\mathbf{v}, \mathbf{w}), \tag{3.12}$$

where $\mathbf{w}$ is a weight vector parameterising f.

Many implementations of scalarisation functions exist, such as the weighted global criterion method, the weighted sum, the Chebyshev scalarisation functions (Dunford et al., 1988) and lexicographic methods, where an order exists among the objectives, similar to how words are ordered in the alphabet. As the preferences of the decision maker are known beforehand, there is no need to approximate the entire Pareto front of optimal solutions. For instance, in Example 3.1, if the decision maker would know its preferred alignment of speed and cost, this information could be used to seed the search process. Since the weight vector $\mathbf{w}$ specifies the compromise solution the user is interested in ahead of time, there is no need to learn a set of trade-off solutions. This way, one can reuse established single-objective solution methods as the dimensionality of the problem is reduced to one. The class of solution techniques that adopt this principle are called *single-policy* algorithms as each execution of the optimisation process yields only a single solution as presented in Figure 3.6. In the upcoming chapters, we will more closely investigate these functions and determine their performance once combined in a reinforcement learning setting.

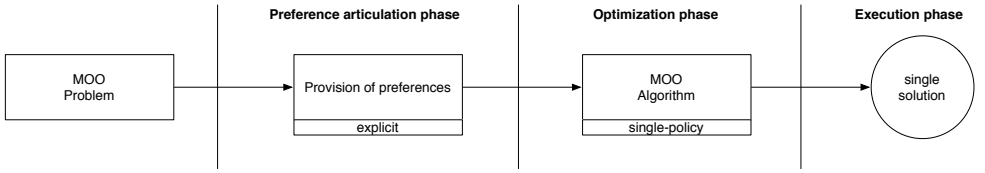


Figure 3.6: In a priori optimisation, the decision maker first supplies its preferences in an explicit format in the articulation phase. In the optimisation process, this explicit preference information is employed in a single-policy algorithm in order to obtain the best compromise solution according to the decision maker.

In some cases, the decision maker might not succeed to express its preferences before the optimisation process is initiated. This could be the case when the decision maker is unfamiliar with the problem at hand or because he might have a hard time explicitly stating its preferences. For instance, in Example 3.1, someone with no experience in automotive vehicles might have a hard time to formally describe its preferences a priori. Another example could be the situation where the government has to decide on where to build a new airport. For this type of problem, the government could quite accurately estimate the monetary costs associated to the build, but it would be much harder to specify a weight vector that evaluates solutions based on other criteria. For example, the increase of noise, traffic and pollution in the area and the social cost of expropriating families from their houses are qualitative rather than quantitative cost functions. In these cases, it is appropriate to offer a palette of trade-off solutions which the decision maker can choose from once he has a clearer view on what solutions appeal to him. The class of algorithms that obtain a set of trade-off solutions rather than a single solution are called *multi-policy* algorithms. How the decision maker finally picks a single solution from this set is defined by a preference articulation process. In this selection process the user can either supply specific weights to define an explicit representation of its preferences or use other, less-quantifiable preferences that allow a more implicit representation. This principle is called a posteriori preference articulation as it makes use of a *generate-first-choose-later* approach (Messac and Mattson, 2002). We depict the process in Figure 3.7.

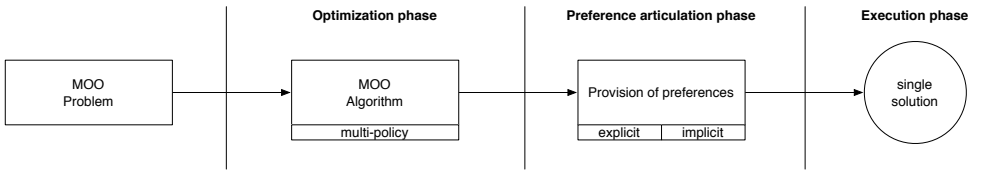


Figure 3.7: In a posteriori optimisation, a multi-policy algorithm first proposes a set of trade-off solutions. In the next step, the decision maker can select a single solution from this set according to its explicit or implicit preferences.

In the two previous approaches, the role of the decision maker was minimal, i.e., he supplied his preference information explicitly or implicitly either before or after the multi-objective optimisation algorithm is initiated. However, the decision maker may also interact with the optimisation process in an iterative manner. For instance, when the set of trade-off solutions obtained by an a posteriori optimisation process is too large or when too many objectives are present, the decision maker might have a hard time choosing its preferred compromise solution. In that case, interactive methods can help to incorporate the possibly subjective and implicit preferences of the decision maker during the optimisation process itself. This interactive setting allows the decision maker to gradually specify its area of interest within the objective space, even if this area was not entirely clear to the decision maker itself at the start of the process. Several approaches are possible to interact with the optimisation process. Firstly, the decision maker can opt to interactively redefine the parameters of the scalarisation function, traditionally used for single-policy methods, to steer the search towards a desired compromise solution (Alves and Climaco, 2000; Sakawa and Kato, 1998; Sakawa and Shibano, 1998). Secondly, instead of evaluating a single solution at a time, the decision maker can also interactively evaluate a finite set of alternative solutions. Each run of the multi-policy algorithm, the decision maker can select the solutions that it finds appealing. The interactive algorithm then takes this partial and limited preference knowledge into account to refine the search to regions in the objective space that match this information. In the end, the decision maker needs to select a single solution in a similar manner as the a posteriori approach to end up with the best trade-off solution (see Figure 3.8). Examples of this principle are the light-beam methods (Jaszkiewicz and Słowiński, 1994, 1995; Kalyanmoy and Abhay, 2007).

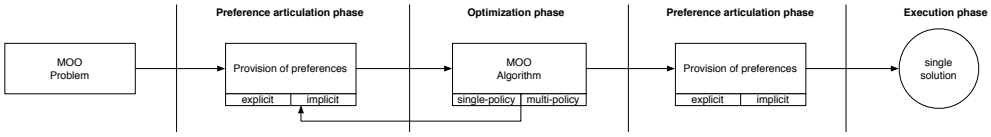


Figure 3.8: In interactive optimisation, the user can first define partial and rough preference information in an implicit or explicit form which the optimisation algorithm uses to suggest a single solution or a set of solutions. In an alternating fashion, the user can redefine and reassign more accurate preference information to steer the search process. In the end, the same principle can be employed to select a single solution for the selection phase.

In this section, we have provided an overview of different scenarios of the decision maker and how preference information influences the search process. We have introduced the two classes of multi-objective optimisation algorithms, called single-policy and multi-policy algorithms. In the following section, we describe some evolutionary multi-objective optimisation algorithms that are particularly suited for multi-policy optimisation.

3.2 Evolutionary Multi-Objective Optimisation

So far, we have seen that multi-objective solution methods can be characterised by whether they learn a single policy or multiple policies at the same time. Single-policy methods may not be very exotic as the function they optimise is usually a scalarisation function applied on the problem's original objective functions. Hence, single-policy techniques relate closely to standard, single-objective solution methods. Moreover, multi-policy algorithms might arouse more excitement as they simultaneously optimise multiple objectives and offer a set of trade-off solutions in a single run. In this regard, *evolutionary algorithms* (EA) are of great interest for multi-policy optimisation as their internal mechanisms are intrinsically coupled to the principle of supplying a set of solutions. In this section, we will more closely elaborate on EAs since they provide foundations to some concepts in multi-objective reinforcement learning.

Evolutionary algorithms are inspired by natural systems and apply the principles of biological evolution to design algorithms for problem solving (Bäck, 1996). Evolutionary algorithms consist of an initially random *population* of individuals that evolves over a series of generations until a termination criterion is met. Each generation, a principle based on the survival of the fittest is mimicked to determine which candidates of the previous generation can seed the next generation based on an *evaluation*, *selection*, *recombination* and *mutation* steps (Spears, 2000). We describe these components in more detail in the following paragraph.

1. In the evaluation step, the current population is evaluated given a certain objective function that assesses its quality.

2. In the selection step, this fitness is used to determine which candidates of the current population are chosen for mating. These candidates will become parents that produce offspring.

3. In the recombination step, the information of two or more selected parents is combined in order to produce offspring.

4. In the mutation step, the offspring undergoes mutation, i.e., small perturbations, in order to maintain genetic diversity from one generation to another.

5. In the reinsertion step, it is determined which elements of the old population and the new offspring are to be inserted in the new population. This step is based on the size of the newly generated offspring and their fitness evaluation.

This process is repeated until the fitness of a candidate solution exceeds a certain threshold or after a predefined number of iterations. An overview of the different steps that comprise an evolutionary algorithm is depicted in Figure 3.9.

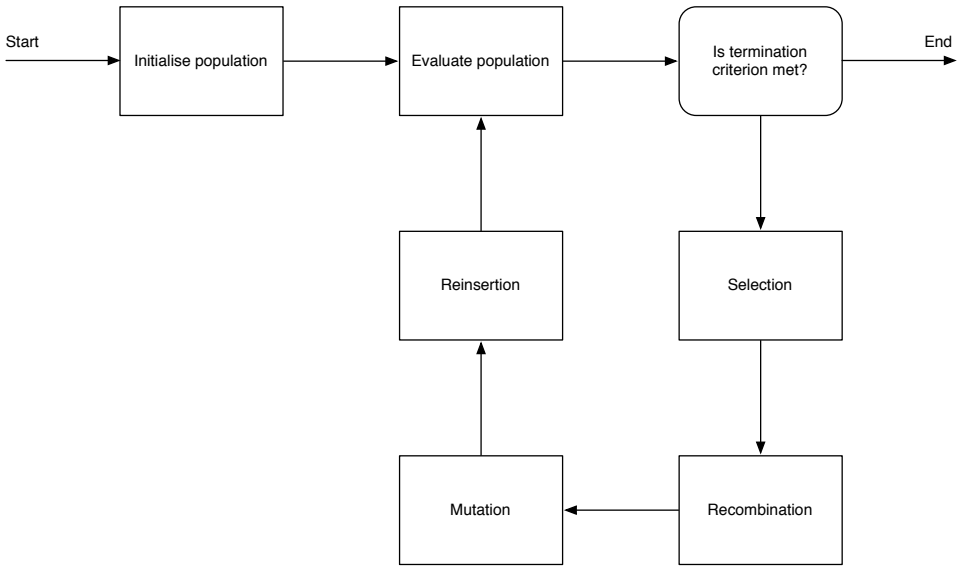


Figure 3.9: An overview of the different components of an evolutionary algorithm.

Due to the fact that evolutionary algorithms evolve a population of solutions rather than a single solution, they are particularly suited for problems involving multiple and conflicting objectives. With this in mind, researchers in the 1980s grew a rapidly increasing interest in the area of evolutionary multi-objective (EMO) algorithms (Schaffer, 1985; Van Veldhuizen and Lamont, 1998; Coello Coello, 1999). As a result of their principal characteristics, EMO algorithms are inherently powerful multi-policy algorithms that yield a set of candidate solutions in a single run.

In this thesis, we focus on sequential decision making under uncertainty. The reasons why we consider reinforcement learning instead of EMO algorithms are as follows:

- Traditionally EMO algorithms are focussed on problems involving a single state. Although workarounds are possible for multi-state environments (Goh and Tan, 2009), they are merely ad-hoc solutions. The reinforcement learning framework is specifically tailored for dealing with multiple states.
- The applications that we believe can benefit from this work in the near future are robotic problems. In these problems, the learning speed and the sampling cost are crucial aspects. Intrinsically, reinforcement learning is concerned with these properties.
- On a theoretical side, the performance and the power of evolutionary algorithms are argued by Holland's schema theorem (Holland, 1992). However, this theorem is not a formal convergence proof. For some reinforcement learning algorithms, such as for instance for the Q-learning algorithm in Section 2.4.2, convergence proofs

exist (Tsitsiklis, 1994). This is why we find it more appropriate to focus on reinforcement learning in this dissertation.

3.3 Definitions of Multi-Objective RL

In the previous section, we have described the theoretical foundations of multi-objective optimisation and all the components that comprise a MCDM problem. In this section, we proceed to the main topic of this dissertation, i.e., multi-objective reinforcement learning (MORL). We characterise the key components that comprise a MORL environment which is a multi-objective Markov decision process. Subsequently, we also categorise the different MORL algorithms in a problem taxonomy. We will analyse two different types of problem taxonomies, i.e., an axiomatic and a utility approach.

3.3.1 Multi-Objective Markov Decision Process

In Section 2.2, we have seen that a Markov decision process is the principal structure to define the environment the reinforcement learning agent is operating in. In the case of a multi-objective problem consisting of m objectives, the environment is modelled as a multi-objective Markov decision process (MOMDP). Formally, a MOMDP is described as follows:

Definition 3.13

A **multi-objective Markov decision process** is a 4-tuple (S, A, T, R), where

- $S = s^1, \ldots, s^N$ is a finite set of states,
- $A = \cup_{s \in S} A(s)$ where $A(s)$ is a finite set of available actions in state $s \in S$,
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition function $T(s'|s, a)$ specifying the probability of arriving in state s' after selecting action a in state s,
- $R : S \times A \times S \rightarrow \mathbb{R}^m$ is the reward function $\mathbf{R}(s, a, s')$ specifying the m-dimensional expected reward vector associated to transitioning from state s with action a to state s'.

From the definition, we can see that the only difference compared to standard MDPs is the reward function. Instead of a single, scalar reward, the reward function now returns an m-dimensional vector in case of a problem with m objectives:

$$\mathbf{R}(s, a, s') = (R_1(s, a, s'), R_2(s, a, s'), \ldots, R_m(s, a, s')) \tag{3.13}$$

The agent is in a certain state of the environment $s(t)$ and has to learn to select the action $a(t)$ at every time step $t = 0, 1, \ldots$. In return, the environment transitions the agent into a new state $s(t + 1)$ and provides a vectorial reward $\mathbf{r}(t + 1)$ representing how beneficial that action was. A schematic overview of the RL agent is given in Figure 3.10.

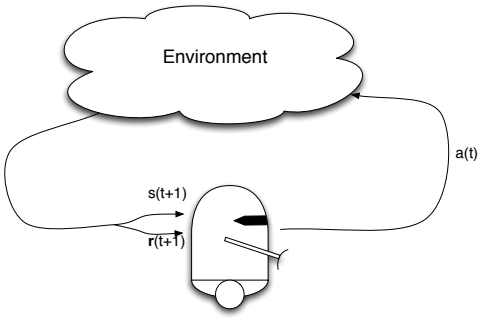


Figure 3.10: The multi-objective reinforcement learning environment returns a vectorial feedback signal $\mathbf{r}(t+1)$ upon the selection of a particular action $a(t)$.

Due to this vectorial reward function, also other reinforcement learning definitions change in the case of multi-objective environments. In the case of a MOMDP, the goal of the agent is to act in such a way that its expected vectorial return, $\mathbf{R}_t$, is Pareto optimal:

$$\mathbf{R}_t = \sum_{k=0}^{\infty} \gamma^k \mathbf{r}_{t+k+1} \tag{3.14}$$

Additionally, the state-dependent value function of a state s is now vectorial:

Definition 3.14

The **value function** $\mathbf{V}^\pi(s)$ specifies how good a certain state s in the long term according to the policy $\pi \in \Pi$. In a MOMDP, the function returns the expected, discounted vectorial return to be observed when the agent would start in state s and follow policy π:

$$\begin{aligned}\mathbf{V}^\pi(s) &= E_\pi\{\mathbf{R}_t | s(t) = s\} \\ &= E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k \mathbf{r}_{t+k+1} \mid s(t) = s \right\}\end{aligned}$$

For the sake of completeness, we review the optimality criterion that we previously defined in Section 3.1.2 now also for multi-objective reinforcement learning. A policy π (strictly) Pareto dominates another policy if it is strictly better on at least one objective of another policy π', while not being strictly worse on the other objectives over all states:

Definition 3.15

A policy $\pi \in \Pi$ **strictly Pareto dominates** another policy $\pi' \in \Pi$, i.e., $\pi' \prec \pi$ when

$$\pi' \prec \pi \iff \exists j : V^{\pi}(s)_j > V^{\pi'}(s)_j \wedge \forall i \neq j : \mathbf{V}^{\pi}(s)_i \not< \mathbf{V}^{\pi'}(s)_i, \forall s \in S. \quad (3.15)$$

A policy π weakly Pareto dominates another policy π', i.e., $\pi' \preceq \pi$ when there does not exist an objective where $\mathbf{V}^{\pi}(s)$ is better than $\mathbf{V}^{\pi}(s')$ over all states.

Definition 3.16

A policy $\pi \in \Pi$ **weakly Pareto dominates** another policy $\pi' \in \Pi$, i.e., $\pi \preceq \pi'$ when

$$\pi' \preceq \pi \iff \forall j : \mathbf{V}^{\pi}(s)_j \geq \mathbf{V}^{\pi'}(s)_j. \quad (3.16)$$

Two policies π and π' are *incomparable* if $\mathbf{V}^{\pi}(s)$ does not Pareto dominate $\mathbf{V}^{\pi'}(s)$ over all states and vice versa:

Definition 3.17

Two policies $\pi \in \Pi$ and $\pi' \in \Pi$ are **incomparable**, i.e., $\pi \parallel \pi'$ when

$$\pi \parallel \pi' \iff \pi \not\prec \pi' \wedge \pi' \not\prec \pi. \quad (3.17)$$

A policy π is *non-dominated* by another policy π' if it is either Pareto dominating or incomparable with that solution:

Definition 3.18

A policy $\pi \in \Pi$ is **non-dominated** by another policy $\pi' \in \Pi$, i.e., $\pi \not\prec \pi'$ when

$$\pi \not\prec \pi' \iff \pi' \prec \pi \vee \pi' \parallel \pi. \quad (3.18)$$

The goal of solving MOMDPs by MORL is to come up with policies that are Pareto optimal, i.e. the policies that are non-dominated by any other policy:

Definition 3.19

A policy $\pi \in \Pi$ is **Pareto optimal** iff it is non-dominated by any other policy $\pi' \in \Pi$:

$$\pi \text{ is Pareto optimal} \iff \forall \pi' \in \Pi : \pi \not\prec \pi'. \quad (3.19)$$

3.4 Problem taxonomy

So far, we have elaborated on the multi-objective reinforcement learning framework and its foundations in multi-objective optimisation. We have discussed three types of preference articulation in multi-criteria decision making and we have presented the notion of scalarisation functions. In this section, we present a problem taxonomy that categorises MORL algorithms according to the type of scalarisation function and the amount of policies learned, i.e., single-policy or multi-policy. Within this taxonomy, we classify related work on MORL and emphasise the main contributions of this dissertation. In literature, two types of problem taxonomies for MORL algorithms exists, i.e., an *axiomatic* and a *utility-based*. We begin by describing the axiomatic approach and emphasise why it is more appropriate in this work than the utility-based approach in Section 3.4.2.

3.4.1 Axiomatic approach

In Figure 3.11, we depict the major MORL algorithms that are published in literature. The approaches in red are the contributions of this dissertation. In the following subsections, we will analyse in more depth each category of related work and briefly outline our improvements to the MORL research field.

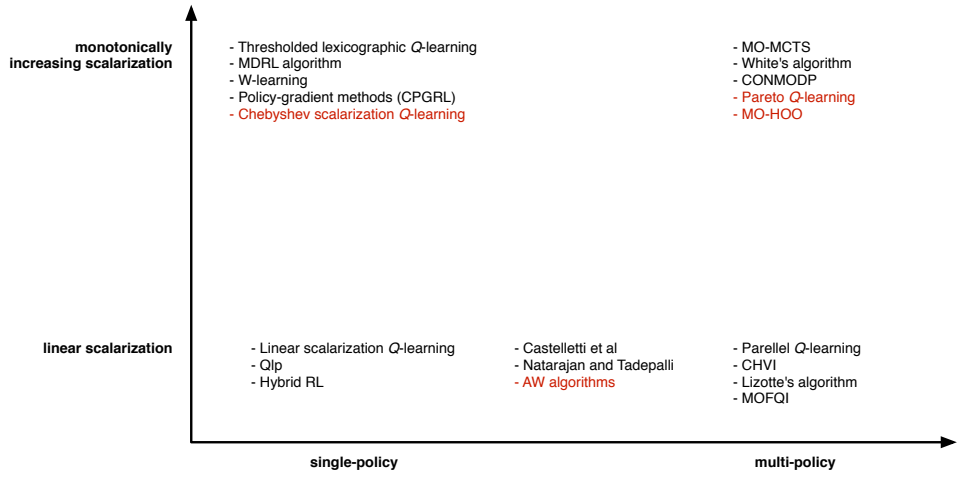


Figure 3.11: An overview of multi-objective algorithms for learning and planning settings. We categorise these approaches on two factors: whether or not they learn a single policy or multiple policies and whether or not the scalarisation function is a linear or a monotonically increasing combination of the objectives. The approaches in red are the contributions of this dissertation.

Single-policy algorithms with a linear scalarisation function

In Section 3.1.2, we have provided a general description of a scalarisation function as a function that projects a vector to a scalar. In the case of MORL, we can make this definition a bit more specific (Roijers et al., 2013):

Definition 3.20

A **scalarisation function** f is a function that projects the multi-objective value $\mathbf{V}^{\pi}$ to a scalar value:

$$V^{\pi}_{\mathbf{w}} = f(\mathbf{V}^{\pi}(s), \mathbf{w}), \tag{3.20}$$

where **w** is a weight vector parameterising f and $\mathbf{V}^{\pi}$ is the value function in state s.

Thus, the scalarisation function maps the multi-objective value function in a state to a scalar value. In this way, a traditional reinforcement learning setting is created as the environment becomes one-dimensional. We differentiate between two types of scalarisation functions. i.e., a *linear* or a *monotonically increasing* scalarisation function. When f computes a linear combinations of the values of a vector given a weight vector $\mathbf{w}$, f is in essence linear. This linear combination is simply a weighted sum of the values for each objective with the corresponding element in $\mathbf{w}$. Following the notation in Roijers et al. (2013), the linear scalarisation function can be formalised by the following definition.

Definition 3.21

A **linear scalarisation function** f computes the inner product of a vector-valued value function $\mathbf{V}^{\pi}$ and a weight vector $\mathbf{w}$

$$V^{\pi}_{\mathbf{w}} = \mathbf{w} \cdot \mathbf{V}^{\pi}, \tag{3.21}$$

where **w** is a positive weight vector parameterising f and $\mathbf{V}^{\pi}$ is the value function in a state s.

Each element of the weight vector $\mathbf{w}$ specifies the relative importance of each objective and should satisfy the following equation:

$$\sum_{o=1}^{m} \mathbf{w}_o = 1. \tag{3.22}$$

The linear scalarisation function possesses some interesting properties. For starters, it is a simple and classical approach that requires few modifications to standard solution methods. Also, it is proven that the linear scalarisation function converges to a Pareto optimal solution (Vamplew et al., 2010). However, the linear scalarisation function also has downsides. For instance, it is not always clear how to specify the weights in order

to find a desired trade-off between solutions. Additionally, as the scalarisation function computes a linear combination of the objectives, only a subset of the Pareto front can be learned as we will see in Chapter 4.

We begin this overview in the simplest category of MORL algorithms, i.e., algorithms that employ a linear scalarisation function to obtain a single policy. These algorithm are particularly tailored for the scenario with an a priori articulation of preferences in Section 3.1.2, where a single policy is to be obtained that is optimal for a given weight vector. Incorporating the linear scalarisation function in MORL is relatively easy as one can apply the scalarisation function on the vectorial reward function to obtain a scalarised feedback signal or one can learn a vectorial value function and apply the scalarisation function in the action selection strategy. The latter is found to be more appropriate in the case of function approximation as the scalarised value might be harder to learn than the values for each of the objectives individually (Roijers et al., 2013; Tesauro et al., 2008). Therefore, learning a single policy through linear scalarisation is not exceptionally difficult as highlighted in Vamplew et al. (2010) where a linear scalarised Q-learning algorithm is proposed that is a trivial extension to the single-objective version of the algorithm.

Although there is no specific need for specialised algorithms for this category, MORL algorithms involving a linear scalarisation function have been an active research topic for many years (Natarajan and Tadepalli, 2005; Castelletti et al., 2010; Vamplew et al., 2010; Roijers et al., 2013). For instance, in Perez et al. (2009), a method is developed for the provision and the allocation of computational resources. The authors use a linear scalarisation function to reduce the dimensionality of the multi-objective feedback signal to obtain a single, scalar reward. In Castelletti et al. (2002), a reinforcement learning algorithm is developed for the operational management of a water system. The solution method is a Q-learning variant, called Qlp, that tries to regulate the provision of water to downstream agricultural users that live near Lake Como in Italy while also providing flood protection on the lake shores. The same authors extended the fitted Q-iteration algorithm or multiple objectives in Castelletti et al. (2010). Fitted Q-iteration is a batch, model-free reinforcement learning algorithm that learns an approximation of the optimal Q-function. Each iteration of the algorithm consists of a single application of the standard Q-learning update rule in Equation 2.8 for each input sample, followed by the execution of a supervised learning method in order to train the next Q-function approximation (Ernst et al., 2005). In Tesauro et al. (2008), the *hybrid RL* algorithm is proposed for balancing power consumption and the response time to user requests in computing servers by linear combinations.

In this dissertation, we analyse the advantages and the limitations of the algorithms of this category in Chapter 4. In the same chapter, we propose a framework for these algorithms, where the scalarisation function can be seen as a parameter. In this framework, the knowledge representation of the agent is extended to store a vector of estimates and the scalarisation function is employed in the action selection process.

Single-policy algorithms with a monotonically increasing scalarisation function

Although the linear scalarisation function is common and its use is widespread (Vamplew et al., 2010), it may not suffice to express every user preference. In that case, non-linear preferences might be more appropriate. Scalarisation functions belonging to this class of algorithms are called *monotonically increasing* scalarisations functions. The set of monotonically increasing functions is a superset of the linear functions, i.e., they are less specific than linear functions and can be defined as follows.

Definition 3.22

A scalarisation function f is monotonically increasing if:

$$\forall i, \mathbf{V}_i^{\pi} \geq \mathbf{V}_i^{\pi'} \implies \forall \mathbf{w}, f(\mathbf{V}^{\pi}, \mathbf{w}) \geq f(\mathbf{V}^{\pi'}, \mathbf{w}) \tag{3.23}$$

where $\mathbf{w}$ is a positive weight vector parameterising f and $\mathbf{V}^{\pi}$ is the value function in a state s.

In layman's terms, this means that the scalarisation function preserves the weakly Pareto dominance relationship. As highlighted in Roijers et al. (2013), monotonically increasing functions are a broad class of functions. For example, the Pareto relation, the Chebyshev function (Dunford et al., 1988) and lexicographic orderings (Gábor. et al., 1998) are members of this class of functions.

Due to specific preference relations of the decision maker, single-policy algorithms with a monotonically increasing scalarisation function also achieved significant attention in MORL research. Humphrys (1997) models a mind with internal tension and competition between selfish behaviours in a decentralised manner by constructing a multi-agent system. To accommodate for intelligent action selection, *W-learning* is proposed. In W-learning different parts of the mind modify their behaviour based on whether or not they are succeeding in getting the body to execute their actions (Humphrys, 1997). The *thresholded lexicographic* Q-learning (TLQ) algorithm is another example of a non-linear scalarisation function (Gábor. et al., 1998). In TLQ, constraints should be met for all but the last objective. These constraints define a minimal threshold level for the values of the objectives, which can be ordered in a lexicographic fashion. For example, in stock markets, the TLQ could find a policy that maximises the return on investment while not exceeding a certain level of risk. While constraints and thresholds are a specific type of preference information, it is also possible to define an area of interest in the objective space as being preferred by the decision maker. This area can then gradually scale and move in an interactive manner as the algorithm proceeds. In Mannor and Shimkin (2002), this idea is proposed in the *multiple directions reinforcement learning* (MDRL) algorithm. MDRL offers a possibility to steer the average reward vector towards a target set using so-called approaching policies. Although these approaching policies employ a standard linear scalarisation function, the mechanism to steer the policies uses a selection strategy that is non-linear. This concept is extended for episodic tasks as well in Vamplew et al. (2009) where stochastic combinations

are constructed of optimal policies that are obtained through linear scalarisation. In Uchibe and Doya (2007), a policy-gradient algorithm is proposed that learns in a direct and iterative manner a single policy satisfying predefined constraints. This algorithm is called *constrained policy gradient reinforcement learning* (CPGRL) as it iteratively updates the policy in the direction of a gradient satisfying one or more constraints. Similar to the previous approaches, mixture policies are constructed with non-linear scalarisation functions.

In Chapter 4, we will more closely analyse monotonically increasing scalarisation functions and their performance in multi-objective reinforcement learning. We argue that the choice of the scalarisation function is crucial since it has a huge impact on the solution the reinforcement learning technique converges to. In this respect, we theoretically and empirically compare the linear and the non-linear *Chebyshev* scalarisation function on their effectiveness in retrieving optimal trade-off solutions.

Multi-policy algorithms with a linear scalarisation function

Multi-policy algorithms that employ a linear scalarisation function aim to learn policies that are optimal for linear combinations of a weight vector and the vector-valued value function. This set of policies represents the *convex hull*, which is a subset of the Pareto front. [2] The convex hull defines a set of policies for which the linear combination of the value of policy π, $\mathbf{V}^{\pi}$, and some weight vector $\mathbf{w}$ is maximal (Roijers et al., 2013).

Definition 3.23

The **convex hull** (CH) of a set of policies $\Pi = \{\pi_1, \ldots, \pi_k\}$ is a set of policies for which there exists a $\mathbf{w}$ which linear combination with the vector-valued value function in a state s is maximal

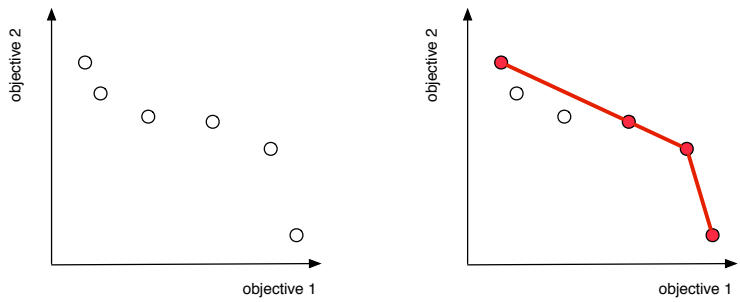
$$CH(\Pi) = \{\pi \in \Pi | \exists \mathbf{w} \in \mathbb{R}^m, \forall \pi' \in \Pi : \mathbf{w} \cdot \mathbf{V}^{\pi}(s) \geq \mathbf{w} \cdot \mathbf{V}^{\pi'}(s)\} \tag{3.24}$$

where **w** is the weight vector satisfying Equation 3.22 and $\mathbf{V}^{\pi}$ is the vectorial value function of policy π in a state s.

In Figure 3.12 (a), white dots denote the Pareto front of a bi-objective problem and in Figure 3.12 (b) the red line represents the corresponding convex hull.

Several algorithms have been proposed that follow this principle. One of the most important contributions is the *convex hull value-iteration* (CHVI) algorithm which computes the deterministic stationary policies that are on the convex hull of the Pareto front (Barrett and Narayanan, 2008). From batch data, CHVI extracts and computes every linear combination of the objectives in order to obtain all deterministic optimal policies. CHVI bootstraps

[2] Do note that in mathematics and geometry, the term 'convex hull' has a different meaning than in MORL. In geometry, the convex hull of a set of points S is the minimal subset of $C \subset S$ so that every point in S can be expressed by a convex combination of the points in C. In MORL, the convex hull is considered the upper envelope of the points in C, i.e., those points whose convex combinations are Pareto dominating all other points in S.



(a) (b)

Figure 3.12: (a) A Pareto front of bi-objective policies represented by white dots. (b) The convex hull of the same Pareto front is represented by a red line.

by calculating the convex hull of the union over all actions in s', that is $\bigcup_{a'} \mathbf{Q}(s', a')$. The most computationally expensive operator is the procedure of combining convex hulls in the bootstrapping rule. The 4 deterministic policies in Figure 3.12 (b), denoted by red dots, are the ones that CHVI would learn.

Lizotte et al. (2010) reduce the asymptotic space and time complexity of the bootstrapping rule by simultaneously learning several value functions corresponding to different weights and by calculating their piecewise linear spline representation. Recently, they validated their work on clinical trial data for three objectives to propose a treatment to patients based on a trade-off between the effectiveness of the drugs and severity of the side effects. Nevertheless, the practical possibilities for higher dimensional spaces are not straightforward (Lizotte et al., 2012).

In Kazuyuki et al. (2009), the *parallel Q-learning* algorithm is proposed. Parallel Q-learning is similar to CHVI in the sense that it learns optimal piecewise-linear policies for all weights. It computes the convex hull by defining sum and union operators for sets, similar to the way CHVI bootstraps. The algorithm suffers from convergence issues in the sense that vertices are continuously being added to the polygon representing the convex hull. The authors introduce a threshold parameter to reduce the size of the sets and to increase the accuracy.

In Castelletti et al. (2011, 2012), the fitted Q-iteration algorithm is extended to learning sets of policies in multi-objective environments. This off-line algorithm is called multi-objective fitted Q-iteration (MOFQI) and it computes the set of expected return vectors obtained for all the possible weight values in a single run.

While the aforementioned multi-policy algorithms learn only a finite set of deterministic policies, it might be interesting to employ probabilistic combinations of these policies. Such a stochastic combination of two policies is called a *mixture policy* (Vamplew et al., 2009). Take for instance a very easy multi-objective problem where the agent can only follow two deterministic policies π_1 and π_2 with $\mathbf{V}^{\pi_1}(s_0) = (1, 0)$ and $\mathbf{V}^{\pi_2}(s_0) = (0, 1)$, where s_0

denotes the start state. If one would follow policy π_1 with probability p and policy π_2 with probability $(1-p)$, the average reward vector would be $(p, 1-p)$. Thus, although there are only two deterministic policies for the original problem, a mixture policy implicates that we can sample the entire convex hull of policies by combining the deterministic policies with a certain probability. Hence, stochastic combinations of the policies of the Pareto front in Figure 3.12 (a) can represent every solution on the red line in Figure 3.12 (b). However, mixture policies might not be appropriate in all situations, as highlighted in Roijers et al. (2013). For instance, in the setting of Lizotte et al. (2010), clinical data is analysed to propose a treatment to patients based on a trade-off between the effectiveness of the drugs and severity of the side effects. Consider the case where only two policies exist that either maximise the effectiveness and the severity of the side effects and vice versa. While the average performance of the mixture policy of these two basic policies might yield good performance across a number of episodes, the policy itself might be unacceptable in each episode individually, i.e., for each patient independently.

Single-policy algorithms for multiple policies In the above subsection, we have discussed on specific and elaborate algorithms that retrieve multiple policies simultaneously. A simpler and straightforward alternative is to run a single-policy algorithm multiple times and to collect the resulting policies in a set. For instance, one can run the linear scalarisation algorithm many times for varying weights to approximate the Pareto front. Nevertheless the approach, being basic and relatively inefficient, is implemented in many settings (Castelletti et al., 2002; Vamplew et al., 2008, 2010). In order to increase the efficiency, a method is also proposed that does not need to learn from scratch but which allows to reuse policies obtained in previous runs of the algorithms (Natarajan and Tadepalli, 2005).

Strictly speaking, this iterative principle is an intermediate approach compared to genuine multi-policy algorithms that learn these policies simultaneously. This is why we choose to place these techniques in between the single and multi-policy setting in Figure 3.11.

In Chapter 5, we analyse how the weight parameters of the linear scalarisation function relate to specific parts of the objective space. We highlight that the mapping from weight space to objective space is non-isomorphic, meaning that it is far from trivial to define an appropriate weight configuration to retrieve a desired trade-off solution. In this light, we propose two adaptive weight algorithms (AWA) that iteratively adjust their weight configurations based on the solutions obtained in previous rounds. These algorithms have the ability to adaptively explore both discrete and continuous Pareto fronts.

Multi-policy algorithm with a monotonically increasing scalarisation function

Because of the properties of linear scalarisation functions, sum and union operators can be quite straightforwardly defined on convex hulls. The downside is that only a subset of the Pareto front will be retrieved, leaving potentially interesting trade-off solutions inaccessible to the decision maker. In White (1982), a dynamic programming (DP) algorithm computes a set of Pareto dominating policies. White's algorithm bootstraps the Pareto dominating Q-vectors of the next state to the set of the current state-action pair. The idea is that,

after the discounted Pareto dominating rewards are propagated and the $\hat{Q}_{set}$'s converge to a set of Pareto dominating policies, the user can traverse the tree of $\hat{Q}_{set}$'s by applying a preference function. As highlighted in Section 2.3, a deterministic stationary policy suffices for single-objective reinforcement learning. In the case of MORL, White (1982) showed that deterministic non-stationary policies, i.e., policies that do not only condition on the current state but usually also on the time step t, can Pareto dominate the best deterministic stationary policies. As a result, in infinite horizon problems with large values for the discount factor, the number of non-stationary policies increases exponentially and therefore it can lead to an *explosion* of the sets. In order to make the algorithm practically applicable, Wiering and de Jong (2007) proposed the CON-MODP algorithm which solves the problem of non-stationary policies by introducing a consistency operator, but their work is limited to deterministic transition functions. Monte Carlo tree search is also extended for multiple objectives in the MO-MCTS method to learn a set of solutions (Wang and Sebag, 2012, 2013). In MCTS, a search tree is incrementally built and explored simultaneously (Coulom, 2007). The nodes of the tree represents visited states and branches represent actions. At the end of an episode, the nodes are weighted according to the outcome of the episode in order to bias the action selection in future plays. In MO-MCTS, the upper confidence bounds of the actions are scalarised in either of two distinct manners. One possibility is to apply the hypervolume quality indicator on the combination of their estimates and the set of Pareto optimal policies computed so far. Because the hypervolume measure is costly to compute, as an alternative, it is also possible to simply determine whether a tree walk obtained a non-dominated reward or not in a boolean fashion. This way, scalarised multi-objective value functions are constructed that ease the process of selecting an action with vectorial estimates.

Up to this moment, there are no on-line reinforcement learning methods for learning multiple policies that lie on the Pareto front. As quoted twice in Roijers et al. (2013), this is a specific need for this type of algorithms although the setting is far from straightforward:

> ... MORL methods that can work with nonlinear scalarisation functions are of substantial importance. Unfortunately, coping with this setting is especially challenging ...
>
> ... we are not aware of any methods that use a value function approach to learn multiple policies on the PCS. When stochastic policies are permitted, the problem is easier [...] However, when only deterministic policies are permitted, the problem is more difficult.

Because of the challenging nature of this setting, a large part of this dissertation focusses on the development of multi-policy methods for learning Pareto optimal policies. To this extent, we propose two algorithms in Chapter 6. These are the *multi-objective hierarchical optimistic optimisation* (MO-HOO) and the *Pareto Q-learning* algorithm.

The MO-HOO algorithm is applicable in multi-objective $\mathcal{X}$-armed bandit problems. This is a class of problems similar to a standard bandit problem where an agent is faced with a continuous finite-dimensional action space instead of a discrete one as the example of

Section 2.5. The goal of the agent is to discover ranges in the continuous action space that provide Pareto optimal rewards. Our algorithm builds a binary tree where each node represents an optimistic estimate of the corresponding area in the objective space. More precisely, each node stores a set of optimistic estimates of the Pareto front attainable from sampling its subtree.

In the Pareto Q-learning (PQL) algorithm, we further refine the idea of the MO-HOO algorithm and extend it for multi-state environments. PQL is the first temporal difference-based multi-policy MORL algorithm that does not use the linear scalarisation function. Internally, Pareto Q-learning incrementally updates sets of vectorial estimates based on the obtained rewards and the set of Pareto non-dominated estimates of the next state. As a result, the algorithm is not limited to the convex hull, but it can learn the entire Pareto front of deterministic non-stationary policies, if enough exploration is provided.

3.4.2 Utility approach

The axiomatic problem taxonomy assumes the Pareto front is the optimality criterion. Recently, a *utility-based* problem taxonomy was proposed where the optimality criterion is not assumed but derived (Roijers et al., 2013). In that taxonomy, it is argued that there are more suitable solution concepts than the Pareto relation to denote the set of optimal policies in MORL. The utility-based taxonomy derives the optimality criterion based on three properties. These properties are (1) the fact if the decision maker requires a single policy or multiple policies, (2) the type of scalarisation function, being linear or monotonically increasing and (3) whether deterministic or stochastic policies are allowed. The utility-based taxonomy is comprehensive and provides justification for the optimality criterion, specialised for the specific context.

In this dissertation, we limit ourselves to deterministic policies as stochastic policies represent a specific type of policies that we do not envisage having a significant potential in real-life engineering applications. Therefore, in the axiomatic taxonomy, we could reduce the utility-based taxonomy by one category. Similar to the utility-based taxonomy, we cluster research based on the number of policies required and the type of scalarisation function. However, we do not agree with the statement in Roijers et al. (2013) that the scalarisation function is publicly stated by the decision maker at the beginning of the MCDM process. In the utility-based approach, they assume the implicit or explicit scalarisation function employed by the decision maker is known and provided beforehand to the optimiser which can then exploit the properties of the scalarisation function at hand. For instance, they assume that in the case of an a posteriori scenario the multi-objective optimiser knows that the decision maker will use a linear scalarisation function in its thought process once the agent has provided a set of solutions. In this regard, the algorithm only needs to retrieve policies on the convex hull as others will be disregarded by the linear scalarisation function once the decision maker discloses its desired weights. In our opinion, the authors then assume there are two articulations of preferences instead of only a single articulation when the weights are provided. We do not agree with the fact that the scalarisation function is always predefined knowledge to the multi-criteria

optimiser or even known by the decision maker before initiating the optimisation process. From our point of view, there should be only a single articulation of preferences where the decision maker reveals its desired scalarisation function and weights at the same time. Therefore, we hold on to the fact that the Pareto front is the only optimality criterion in the general MCDM case. In the case where the scalarisation function can be assumed before initiating the process, a utility-based taxonomy could be employed.

3.5 Summary

In this chapter, we introduced general concepts for learning in multi-criteria environments. We have defined multi-criteria decision making, the umbrella term for describing a collection of approaches which seek to take explicit account of the multiple objectives in order to facilitate and improve decision making. Multi-objective optimisation is an important branch of MCDM which solves these problems by means of mathematical equations. Generally speaking, by solving MCDM problems, one needs to find solutions that are Pareto optimal. Because this optimality criterion entails a partial order, there usually exists no single optimum solution as in the case of single-objective optimisation. In this case, there exists a set of optimal trade-off solutions, called the Pareto front. We have drawn attention to the role of the decision maker in the MCDM process and how it can articulate its preferences. We have highlighted three different types of preference articulation which influence the flow of the optimisation process and how the decision maker and the multi-criteria solver interact.

Evolutionary multi-objective optimisation is an important class of algorithms that are particularly suited for multi-objective problems as they evolve a set of solutions in a single run. These techniques are intrinsically multi-policy algorithms as they obtain a set of solutions instead of a single solution, as single-policy algorithms do.

After carefully introducing the foundations of multi-objective optimisation, we focussed on the main topic of this dissertation, which is multi-objective reinforcement learning. We have defined multi-objective Markov decision processes and what constitutes an optimal policy. In the end, a problem taxonomy is provided which classifies MORL research on 4 categories, i.e., the number of policies obtained (single or multi-policy) and whether the scalarisation is linear or monotonically increasing. In the following chapter, we will more closely analyse different types of scalarisation functions for MORL and to what degree they can be used to approximate the set of optimal trade-off solutions.

4 | Learning a Single Multi-Objective Policy

Multi-objective reinforcement learning algorithms that employ a scalarisation function to obtain a single multi-objective policy have been omnipresent in research (Gábor. et al., 1998; Vamplew et al., 2008; Perez et al., 2009; Van Moffaert et al., 2013b). Most commonly, a linear scalarisation function is used to transform the multi-objective nature of the problem to a standard single-objective RL problem (Castelletti et al., 2002; Tesauro et al., 2008), although non-linear scalarisation functions could potentially solve a few of its limitations (Roijers et al., 2013). In this chapter, we will both theoretically and empirically analyse the linear scalarisation function and the non-linear Chebyshev scalarisation function. We will elaborate on the advantages and the limitations of each scalarisation function and highlight how these scalarisation functions can be incorporated in a general framework.

In more detail, we will present the following contributions in this chapter:

- Present a general framework for scalarised MORL
- Investigate the usefulness of the linear scalarisation function and its limitations
- Research on a theoretical and empirical level whether the Chebyshev scalarisation function can also be applied in MORL

This research has been published in Van Moffaert et al. (2013b,a,c).

4.1 A framework for scalarised MORL algorithms

Before we analyse the specifics of linear and non-linear scalarisation functions, we expand on how the higher dimensionality of the reward signal influences the knowledge representation of the agent. Traditionally, in single-objective reinforcement learning, the agent keeps

a table of $Q(s, a)$-values that store the expected cumulative discounted reward for the combination of each state s and action a in the environment. These $Q(s, a)$-values are then updated over time using an update rule, such as for instance the traditional Q-learning update rule of Equation 2.8.

In multi-objective environments, the knowledge representation of the agent is not abundant enough to store the supplementary information the environment provides. In the case of learning a single multi-objective policy, the agent now stores a $\hat{\mathbf{Q}}$-vector for each state-action pair.[1] On its turn, a $\hat{\mathbf{Q}}$-vector comprises a $\hat{Q}$-value for each objective, i.e.,

$$\hat{\mathbf{Q}}(s, a) = (\hat{Q}_1(s, a), \ldots, \hat{Q}_m(s, a)). \tag{4.1}$$

These $\hat{\mathbf{Q}}$-vectors are then updated for each component individually. For instance, in the case of multi-objective Q-learning, the bootstrapping rule for objective o of a $\hat{\mathbf{Q}}$-vector is:

$$\hat{Q}_o(s, a) \leftarrow \hat{Q}_o(s, a) + \alpha_t(\mathbf{r}_o + \gamma \max_{f(\hat{\mathbf{Q}}(s', a'), \mathbf{w})} \hat{Q}_o(s', a') - \hat{Q}_o(s, a)), \tag{4.2}$$

where $\mathbf{r}_o$ is the o'th component of the immediate reward vector and $\max_{f(\hat{\mathbf{Q}}(s', a'), \mathbf{w})} \hat{Q}_o(s', a')$ retrieves the o'th component of the $\hat{\mathbf{Q}}$-vector that contains the maximum scalarised value for a given scalarisation function f and a weight vector $\mathbf{w}$.

In on-line learning, an action strategy, such as ϵ-greedy and softmax, needs to be employed to decide on the next action. When learning a single multi-objective policy, the scalarisation function f is adopted as a scoring mechanism for action selection strategies. The function transforms a multi-dimensional $\hat{\mathbf{Q}}$-vector into a scalar value that comprises a combined score for an action a based on its different objectives. We refer to the $\widehat{SQ}(s, a)$ to denote the scalar outcome of applying f on $\hat{\mathbf{Q}}(s, a)$. Therefore, when having to select an action in a state s, the $\widehat{SQ}(s, a)$-values for each action a in s are computed and passed on to the traditional action selection strategy. In the case of a scalarised ϵ-greedy selection strategy, the pseudo code is depicted in Algorithm 2. For the current state s, the agent initialises a list named $SQList$ to store the scalar $\hat{Q}$-values. For each action in s, we compute the scalar $\widehat{SQ}(s, a)$-value and append it to the $SQList$, which the ϵ-greedy strategy uses to make its decision on the next action to select.

The scalarised multi-objective Q-learning algorithm comprises the main aspects highlighted above, i.e., the multi-objective bootstrapping rule and a scalarised action selection strategy. An outline of the algorithm is listed in Algorithm 3. At line 1, the $\hat{Q}$-values for each triplet of states, actions and objectives are initialised. The agent starts each episode in state s (line 3) and chooses an action based on the multi-objective action selection strategy at line 5, e.g, scalarised ϵ-greedy. Upon taking action a, the environment transitions the agent into the new state s' and provides the vector of sampled rewards $\mathbf{r}$. As the

[1] It is also possible to directly scalarise the reward signal and learn standard $\hat{Q}(s, a)$-values instead. In literature, this is called expectation of scalarised return, while learning a $\hat{Q}_o(s, a)$, as is most common in literature, is called scalarisation of expected return. Although both approaches seem identical, there are examples where different policies are optimal for each of the methods. For more information on both principles, we refer to (Roijers et al., 2013)

Algorithm 2 Scalarised ϵ-greedy strategy

1: $SQList \leftarrow \{\}$
2: **for** each action $a_i \in A$ **do**
3: $\quad \widehat{SQ}(s,a) \leftarrow f(\hat{\mathbf{Q}}(s,a), \mathbf{w})$ ▷ Scalarise $\hat{\mathbf{Q}}(s,a)$-values
4: $\quad$ Append $\widehat{SQ}(s,a)$ to $SQList$
5: **end for**
6: **return** ϵ-greedy($SQList$)

Q-table has been expanded to incorporate a separate value for each objective, these values are updated for each objective individually using the modified Q-learning update rule at line 8. More precisely, the $\hat{Q}$-values for each triplet of state s, action a and objective o are updated using the corresponding reward for each objective, $\mathbf{r}$, into the direction of the best scalarised action of the next state s'. It is important to note that this framework only adds a scalarisation layer on top of the action selection mechanisms of standard reinforcement learning algorithms.

Algorithm 3 Scalarised multi-objective Q-learning algorithm

1: Initialise $\hat{\mathbf{Q}}(s,a)$ arbitrarily
2: **for** each episode t **do**
3: $\quad$ Initialise state s
4: $\quad$ **repeat**
5: $\quad\quad$ Choose action a from s using a policy derived from $\hat{\mathbf{Q}}$, e.g., scalarised ϵ-greedy
6: $\quad\quad$ Take action a and observe state $s' \in S$ and reward vector $\mathbf{r} \in \mathbb{R}^m$
7: $\quad\quad$ **for** each objective o **do**
8: $\quad\quad\quad \hat{Q}_o(s,a) \leftarrow \hat{Q}_o(s,a) + \alpha_t(\mathbf{r}_o + \gamma \max_{f(\hat{\mathbf{Q}}(s',a'),\mathbf{w})} \hat{Q}_o(s',a') - \hat{Q}_o(s,a))$
9: $\quad\quad$ **end for**
10:
11: $\quad\quad s \leftarrow s'$ ▷ Proceed to next state
12: $\quad$ **until** s is terminal
13: **end for**

4.2 Linear scalarised MORL

After introducing a general framework for scalarised MORL algorithms, we can proceed by specifying the scalarisation function f. When f computes a linear combination of the values of a vector given a weight vector $\mathbf{w}$, f is linear in nature. This function then computes the dot product of the $\hat{\mathbf{Q}}$-vector of a state-action pair and a corresponding

non-negative weight vector:

$$\begin{aligned}\widehat{SQ}(s,a) &= \hat{\mathbf{Q}}(s,a)\cdot\mathbf{w}\\ &= \sum_{o=1}^{m}\hat{\mathbf{Q}}_o(s,a)\cdot\mathbf{w}_o,\end{aligned} \tag{4.3}$$

where $\mathbf{w}$ is a weight vector specifying the relative importance of each objective. The weight vector itself should satisfy the following equations:

$$\forall o \in [1,m] : 0 \leq \mathbf{w}_o \leq 1. \tag{4.4}$$

$$\sum_{o=1}^{m}\mathbf{w}_o = 1. \tag{4.5}$$

4.2.1 Optimality of linear scalarised MORL

One important advantage of the linear scalarisation function is its property of additivity. In essence, this means that the result of the scalarisation function applied on the vectorial value function of a policy π is equal to applying the scalarisation function in the learning process directly:

$$\begin{aligned}f(\mathbf{V}^{\pi}(s),\mathbf{w}) &= \mathbf{V}^{\pi}(s)\cdot\mathbf{w}\\ &= E_{\pi}\{\mathbf{R}_t|s(t)=s\}\cdot\mathbf{w}\\ &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}\mathbf{r}_{t+k+1}\mid s(t)=s\right\}\cdot\mathbf{w}\\ &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}\mathbf{w}\cdot\mathbf{r}_{t+k+1}\mid s(t)=s\right\}\\ &= E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}f(\mathbf{r}_{t+k+1},\mathbf{w})\mid s(t)=s\right\}\end{aligned} \tag{4.6}$$

Hence, the single-objective MDP that was obtained by applying the linear scalarisation function on the original MDP has additive returns. As a result, the linear scalarisation function in combination with the Bellman equation, and thus RL in general, is proven to converge to the optimal policy π^* for the weight vector $\mathbf{w}$. Subsequently, this optimal policy is also optimal in the original objective space:

Theorem 4.1

If f is a linear scalarisation function and $\mathbf{V}^{\pi^*}$ is the vector-valued value function of the converged and optimal policy π^*, then π^* is also Pareto optimal.

$$\pi \in \Pi \wedge \forall\, \pi' \in \Pi : \mathbf{V}^{\pi}\cdot\mathbf{w} \geq \mathbf{V}^{\pi'}\cdot\mathbf{w} \implies \pi \text{ is Pareto optimal.}$$

For insights in the proof of this theorem, we refer to Nakayama et al. (2009).

4.2.2 Convexity of linear scalarised MORL

So far, we have seen that the linear scalarisation function is theoretically proven to converge to a Pareto optimal solution in the objective space of the original problem. In this section, we will analyse whether the inverse also holds, i.e., is every Pareto optimal solution discoverable by employing a linear scalarisation function or not?

In Das and Dennis (1997), this research question is answered by examining the influence of the weights in the objective space. For simplicity reasons, the authors analysed the matter for a problem with two objectives f_1 and f_2, but it could be generalised for m-objective environments as well. The authors argue that the weight space of the linear scalarisation can be considered a goniometric circle with a radius of 1 in order to satisfy Equation 4.5. A visual representation of the process is provided in Figure 4.1 where the Pareto front is entirely convex. The weights of the two objectives can then be represented by $w_1 \in [0, 1]$ and $w_2 = 1 - w_1$. The goal of the optimisation process is then to find the solution $\mathbf{x} \in \mathbf{X}$ so that:

$$\max_{\mathbf{x} \in \mathbf{X}} w_1 f_1(\mathbf{x}) + (1 - w_1) f_2(\mathbf{x}). \tag{4.7}$$

In the goniometric circle, this equation can be rewritten by considering the angle $\theta \in [0, \frac{\pi}{2}]$:

$$\max_{\mathbf{x} \in \mathbf{X}} \frac{\sin\theta}{\sin\theta + \cos\theta} f_1(\mathbf{x}) + \frac{\cos\theta}{\sin\theta + \cos\theta} f_2(\mathbf{x}). \tag{4.8}$$

The utilisation of the weights can be seen as a transformation of the original axis of each of the objectives, i.e., f_1 and f_2 to f_1 and f_2 in Figure 4.1. The rotated f_1 and f_2 axes can be found by an elementary coordinate transformation:

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{4.9}$$

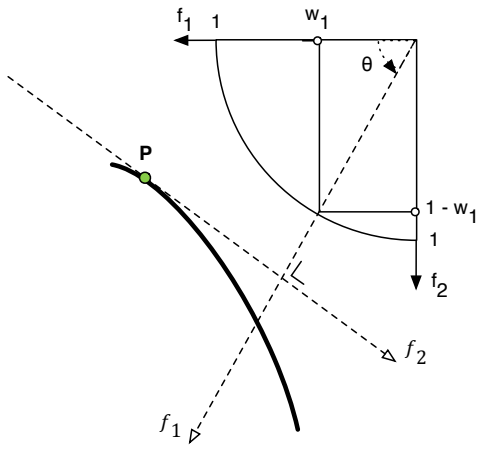


Figure 4.1: The linear scalarisation function can find any solution on the Pareto front with an appropriate weight vector if it the Pareto front has a convex shape.

Therefore, the goal is to find the solution $\max_{x \in \mathbf{X}} f_1$, with $f_1 = f_1 \cos\theta + f_2 \sin\theta$. In Figure 4.1, this maximisation can be regarded as a parallel translation of f_2 to itself until it hits the curve of the Pareto front (Das and Dennis, 1997). In this case, the point P is the solution that is discovered by the linear scalarisation function employing a weight vector of $(w_1, 1 - w_1)$ as f_2 is tangent to the Pareto front at that point. Since the Pareto front in this example is convex, it is equal to the convex hull and every element of the Pareto front can be discovered by varying the θ angle, and thus the weight vector.

Theorem 4.2

If the value function $\mathbf{V}^\pi$ of a policy π lies on the convex hull of the Pareto front, then there exists a weight vector $\mathbf{w}$ for which $\mathbf{V}^\pi \cdot \mathbf{w}$ is maximal:

$$\pi \in CH(\Pi) \implies \exists \mathbf{w} \in \mathbb{R}^m \wedge \forall \pi' \in \Pi : \mathbf{V}^\pi \cdot \mathbf{w} \geq \mathbf{V}^{\pi'} \cdot \mathbf{w}.$$

The next question to ask is whether this property holds for every possible shape of the Pareto front, as we have discussed in Section 3.1.2. In Figure 4.2, we continue the theoretical analysis of the linear scalarisation function on a combination of a convex and a non-convex Pareto front. In the figure, we see that the Pareto front is convex on the left side of point P and on the right side of point Q. In between those two points, the Pareto front is bulging inwards and therefore represents a non-convex area. Given a weight vector $(w_1, 1 - w_1)$, we see that f_2 hits the Pareto front at two points P and Q simultaneously. This causes no issues in the above process, since f_2 is still tangent to the Pareto front, only in this case, at two points. Therefore, points P and Q are discoverable by the linear scalarisation function.

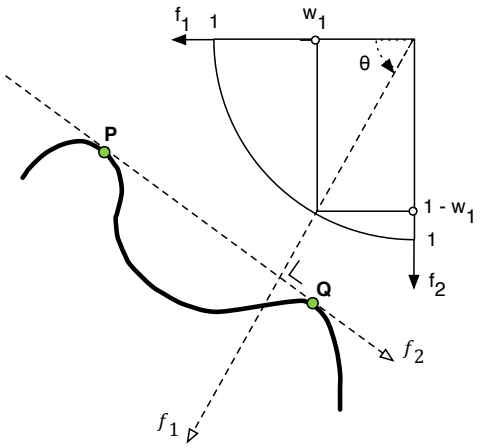


Figure 4.2: In the case the orthogonal axis f_2 is tangent to the Pareto front at two distinct points at the same time, both points can still be discovered with a linear scalarisation function.

The problem arises when one is interested in observing the solutions between points P and Q. In Figure 4.3, we see that point R, located in the non-convex part of the Pareto front, hits the tangent f_2' with the weight vector $(w_{1'}, 1 - w_{1'})$. However, we note a problem as f_2' also intersects at another point S, where f_2' is not tangent to the Pareto front. Based on the continuity and differentiability of the Pareto front, Das and Dennis (1997) analysed that if this characterisation holds, the solution R cannot be found by a linear scalarisation function, regardless of the weight vector. This conclusion also holds for other points in the non-convex area between points P and Q.

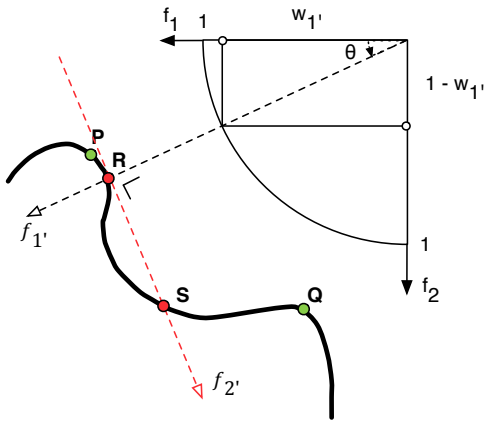


Figure 4.3: Solutions located in the non-convex area of the Pareto front cannot be retrieved by a linear scalarisation function.

Experimental validation

The theoretical properties of the linear scalarisation function have provided us insights in its abilities and limitations. However, this theoretical analysis has been conducted for optimisation processes in general and not for multi-objective reinforcement learning in particular. Therefore, we will examine this scalarisation function methodologically and in detail on two small multi-objective Markov decision processes. These two environments contain a convex and non-convex Pareto front, respectively, and therefore provide a good test bed to interpret the characteristics of the scalarisation function.

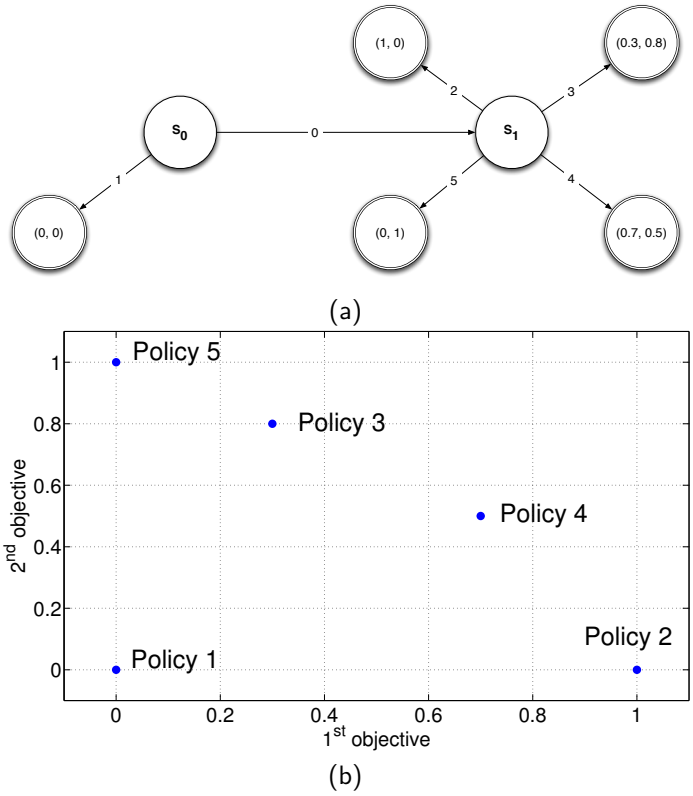


Figure 4.4: A multi-objective Markov decision processes (a) with a convex Pareto front (b).

Before we analyse the results, we briefly introduce the two environments. The environments contain each 7 states and 6 actions, as presented in Figures 4.4 (a) and 4.5 (a). Each episode, the agent starts in the state S_0 and aims to reach one of the 5 goal locations, denoted by a double border. Each time the agent selects an action, the environment deterministically transfers the agent to the corresponding subsequent state and returns a deterministic bi-objective reward signal. The reward vector associated with each action, is depicted in the cell of the corresponding succeeding state. In Figure 4.4 (b) and Figure 4.5 (b), we illustrate the Pareto front of both environments. In each world, the corresponding Pareto optimal set contains the policies ending with actions $2, 3, 4$ and 5, while the policy that selects 1 in the beginning of the episode is the dominated action. Depending on the value of the vector of each action, the Pareto front is either entirely convex or non-convex.

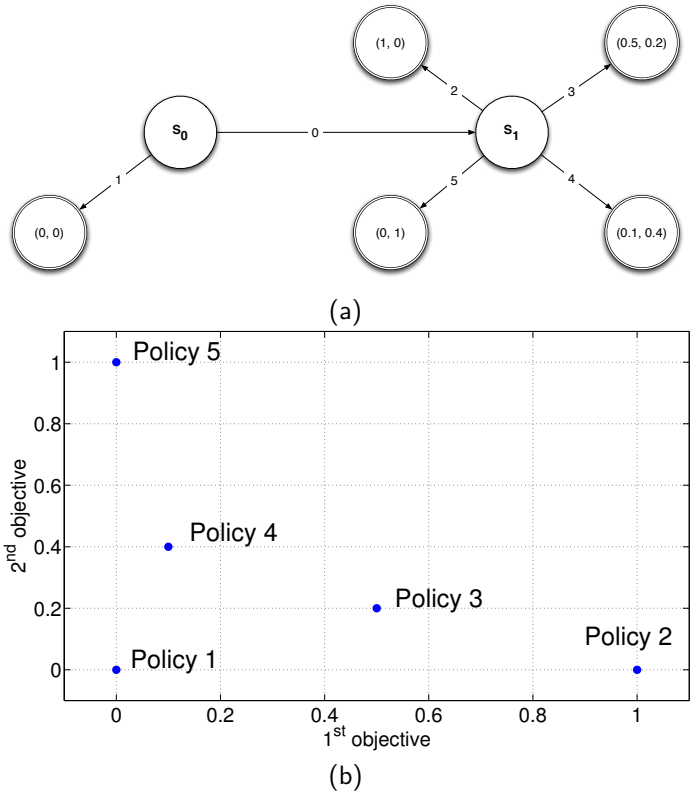
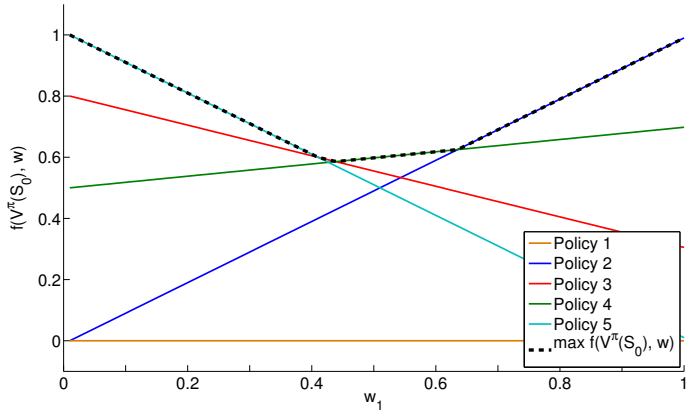
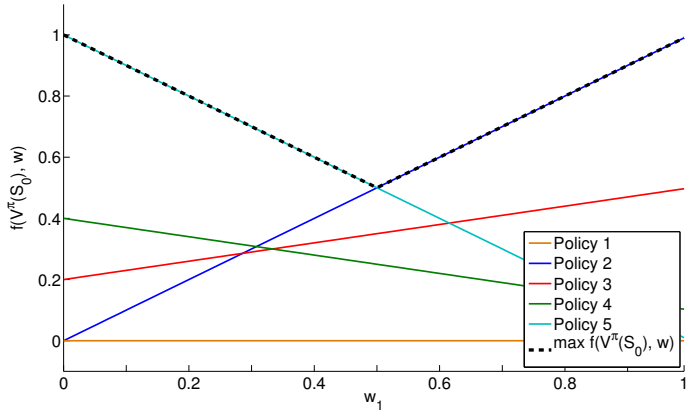


Figure 4.5: A multi-objective Markov decision processes (a) with a non-convex Pareto front (b).

In Figures 4.6 (a) and (b), we depict the outcome of linear scalarised MORL algorithm on both environments. The results were obtained after learning the $\hat{\mathbf{Q}}$-vectors of each state-action pair until the bootstrapping rule in Equation 4.2 did not incur any updates larger than a small Δ to the value of each objective.



(a)



(b)

Figure 4.6: The optimal actions for each weight vector using the linear scalarisation function on environments with a convex and a non-convex Pareto front, respectively.

For the environment with the convex Pareto front in Figure 4.6 (a), we depict a list of linear functions, one for each policy. Each of these lines can be represented by a list of trade-offs, i.e., different values of $w_1 \in [0, 1]$ while $w_2 = 1 - w_1$, together with a list of their corresponding scalarised value function values from the start state S_0, i.e., $f(\mathbf{V}^\pi(S_0), \mathbf{w})$. The set of policies that are optimal for some $w_1 \in [0, 1]$ are the policies whose line representation lies on the upper convex envelope of the scalarised value functions (Lizotte et al., 2010). This representation allows us to discover which policies are optimal for which weight vector, i.e., the policies that lie on the black line in Figure 4.6 (a). In this figure, we see that policy 1 is never optimal as its line does not intersect with the black line. We also note that polices $2, 4$ and 5 are optimal for a relatively large portion of the weight space. Although, it is hardly noticeable on the figure, the linear scalarised MORL

algorithm also successfully identified policy 3 as being optimal, albeit only for a very small part of the weight space. These results correspond to the Pareto front in Figure 4.4 (b) and, therefore, they also confirm that the linear scalarisation function can discover every element on the convex hull.

For the non-convex Pareto front, we highlight the results in Figure 4.6 (b). Although we know that four policies in this environment are Pareto optimal, the linear scalarised MORL algorithm has only identified two regions of the weight space where two policies are optimal. In this non-convex environment, the scalarised value functions of the other policies are always lower than the scalarised value functions of the policies with returns $(1, 0)$ and $(0, 1)$. The linear scalarised MORL algorithm is only able to discover the *extremes* of the non-convex Pareto front, i.e., the knots in the Pareto front where the curve starts to bulge inwards. Depending on the weight vector, either of these extreme policies are retrieved, leaving the other, also Pareto optimal, policies undiscovered.

We can conclude the section on the linear scalarisation function by emphasising its simplicity and usability in MORL. We have seen it is theoretically proven to converge to a Pareto optimal policy. However, only policies on the convex hull, a subset of the Pareto front, can be discovered, leaving the optimal solutions in the concavities of the Pareto front undiscovered. In the next section, we will both theoretically and empirically analyse an alternative scalarisation function that overcomes the limitations the linear scalarisation function imposes.

4.3 Chebyshev scalarised MORL

So far we have seen that the linear scalarisation function fits well into the reinforcement learning framework as it preserves additivity. As a result, the linear scalarisation function in combination with MORL is guaranteed to converge to a policy on the convex hull of the Pareto front (Das and Dennis, 1997). Nevertheless, the convex hull is only a subset of the Pareto front which might not match the preferences of the decision maker. In order to account for every possible preference the decision maker might yield, we are in need for methods that are able to discover the entire set of optimal trade-off solutions. To overcome these limitations, researchers have been focussing on non-linear scalarisation functions (Humphrys, 1997; Mannor and Shimkin, 2002). Some successful approaches within this field employ a thresholded lexicographic order on the objectives (Gábor. et al., 1998; Issabekov and Vamplew, 2012). In these approaches, a minimal threshold level should be met for the values of each of the objectives, which can then be ordered in a lexicographic order. Nonetheless, the scalarisation functions in these algorithms are still very specific and tuned to the problem at hand. More precisely, they lack simplicity in their formulation which is one of the main contributions to the success of the linear scalarisation function. In this prospect, we investigate an alternative non-linear scalarisation function which has proven its efficiency and effectiveness in the general multi-objective optimisation

domain. This scalarisation function is the *Chebyshev* scalarisation function, named after the Russian mathematician Pafnuty Lvovich Chebyshev.[2]

4.3.1 Formal definition

The Chebyshev scalarisation function is an alternative scalarisation function based on L_p distance metrics (Dunford et al., 1988). These metrics measure the distance between two vectors $\mathbf{a} = (a_1, \ldots, a_m)$ and $\mathbf{b} = (b_1, \ldots, b_m)$, where m is the dimensionality of the vector space. The L_p distance between points $\mathbf{a}$ and $\mathbf{b}$ is then defined as:

$$\begin{aligned} L_p(\mathbf{a}, \mathbf{b}) &= \|\mathbf{a} - \mathbf{b}\|_p \\ &= \big(\sum_{o=1}^{m} |\mathbf{a}_o - \mathbf{b}_o|^p\big)^{1/p}. \end{aligned} \tag{4.10}$$

In the case where $p = 1$, the distance metric is:

$$\begin{aligned} L_1(\mathbf{a}, \mathbf{b}) &= \|\mathbf{a} - \mathbf{b}\|_1 \\ &= \sum_{o=1}^{m} |\mathbf{a}_o - \mathbf{b}_o|. \end{aligned} \tag{4.11}$$

This distance metric calculates the sum of the absolute differences of the coordinates of each vector. This principle alludes to the distance one would need to walk in a big city to reach point $\mathbf{b}$ starting from point $\mathbf{a}$, since one cannot cross through buildings but needs to stay on the streets. Therefore, it is also known as the *taxicab* or *Manhattan* distance.The most common L_p metric is the Euclidean distance with $p = 2$. The Euclidean distance is defined as the square root of the sum of the squares of the differences between the corresponding coordinates of the points a and b.

$$\begin{aligned} L_2(\mathbf{a}, \mathbf{b}) &= \|\mathbf{a} - \mathbf{b}\|_2 \\ &= \sqrt{\sum_{o=1}^{m} (\mathbf{a}_o - \mathbf{b}_o)^2}. \end{aligned} \tag{4.12}$$

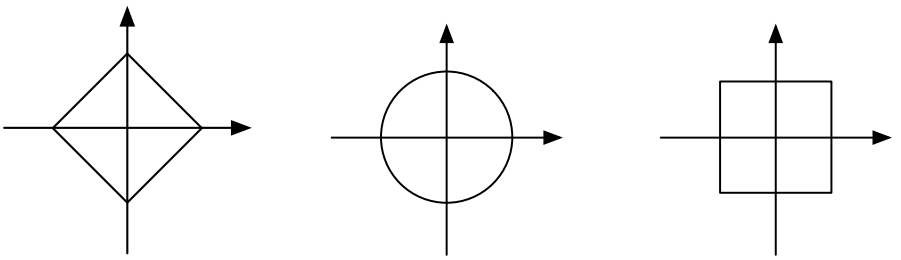
In the case of $p = \infty$, the metric results in the L_∞ or the Chebyshev metric. The Chebyshev metric is defined as the greatest distance between $\mathbf{a}$ and $\mathbf{b}$ along any dimension (Xu et al., 2013). Its mathematical equation is defined as:

$$\begin{aligned} L_\infty(\mathbf{a}, \mathbf{b}) &= \|\mathbf{a} - \mathbf{b}\|_\infty \\ &= \max_{o=1\ldots m} |\mathbf{a}_o - \mathbf{b}_o|. \end{aligned} \tag{4.13}$$

The best way to compare the different distance metrics is to consider their *unit spheres*. A unit sphere is a set of points that lies at a distance of one from a fixed central point.

[2] His name can be alternatively transliterated as Chebychev, Chebysheff, Chebyshov, Tchebychev, Tchebycheff, Tschebyschev, Tschebyschef or Tschebyscheff

Depending on the distance metric used, this sphere will be of a particular shape. Below, in Figure 4.7 (a), (b) and (c), one can find and compare the unit spheres for the Manhattan, Euclidean and Chebyshev distance metrics, respectively.



(a) (b) (c)

Figure 4.7: The unit spheres of the L_1, L_2 and L_∞ distance metrics.

From the figure, we see that each shape intersects each axis at distance one from the central point. We also note that the smaller the p-value, the smaller the size of the unit sphere.

4.3.2 Scalarisation with the Chebyshev function

Now that we have properly introduced the L_p distance metrics, we research how they can be used as a scalarisation function in multi-objective optimisation. The observant reader might have already noticed that the L_1 distance metric is very similar to the equation of the linear scalarisation function. Actually, when one introduces weight vectors in Equation 4.11 and when $\mathbf{b} = (0, \ldots, 0)$, the function becomes:

$$\begin{aligned} L_1(\mathbf{a}, \mathbf{b}) &= \mathbf{w}_1|\mathbf{a}_1 - 0| + \ldots + \mathbf{w}_m|\mathbf{a}_m - 0| \\ &= \mathbf{w}_1|\mathbf{a}_1| + \ldots + \mathbf{w}_m|\mathbf{a}_m| \\ &= \mathbf{w} \cdot \mathbf{a}, \end{aligned} \tag{4.14}$$

which is equivalent to the formula of the linear scalarisation function if the necessary constraints of Equation 4.4 and 4.5 are satisfied for $\mathbf{w}$.

The Chebyshev distance metric can also be transformed to serve as a scalarisation function in multi-objective optimisation (Bowman and Joseph, 1976). In these approaches, the Chebyshev norm is used to scalarise a given vector $\mathbf{a}$ by calculating the Chebyshev distance from that point to an attraction point $\mathbf{z}^* \in \mathbb{R}^m$. This attraction point serves as a point of interest to guide the vector $\mathbf{a}$ to certain areas of the objective space. In Steuer and Choo (1983), this idea is pursued by interactively redefining the attraction point to attain certain solutions in the objective space. In more traditional approaches, the attraction point acts as an ideal point, as described in Section 3.1.2. In this case, $\mathbf{z}^*$ is a parameter

that is constantly being adjusted during the optimisation process by recording the best value so far for each objective o, plus a small negative or positive constant τ, depending whether the problem is to be minimised or maximised, respectively (Klamroth and Jørgen, 2007; Dächert et al., 2012).

One of the main advantages of the Chebyshev scalarisation function in general multi-objective optimisation is that every Pareto optimal solution can be discovered by appropriately varying the weights and/or the attraction point. For a formal proof, we refer to Miettinen (1999). Nevertheless, it remains hard to specify appropriate values for the $\mathbf{w}$ and $\mathbf{z}^*$ parameters in order to discover a desired trade-off solution (Marler and Arora, 2004).

4.3.3 Convergence issues in MORL

In the previous sections, we have introduced the foundations of the Chebyshev scalarisation function and its ability to discover every Pareto optimal solution. This property is theoretically proven to hold in the general topic of multi-objective optimisation. In this section, we will verify whether this property also holds in MORL.

Before we can test the assumption, we need to specify how we can incorporate the Chebyshev function in the scalarised MORL framework of Section 4.1. To accommodate for a new scalarisation function, we only need to specify how one calculates the scalarised $\widehat{SQ}(s,a)$-value given the $\mathbf{Q}$-vector of that same state-action pair. Utilising the Chebyshev function in this process gives us:

$$\widehat{SQ}(s,a) = -\max_{o=1\ldots m} \mathbf{w}_o \cdot |\hat{\mathbf{Q}}_o(s,a) - \mathbf{z}^*_o|, \tag{4.15}$$

with $\mathbf{w}$ satisfying both Equation 4.4 and 3.22. Note that the Chebyshev formula is negated to account for a maximisation problem.

However, for a scalarisation function to be suitable in reinforcement learning, it needs to preserve additivity. Therefore, for any choice of $\mathbf{w} \in \mathbb{R}^m$ and $\mathbf{z}^* \in \mathbb{R}^m$, it needs to hold that

$$f(\mathbf{V}^\pi, \mathbf{w}) = E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k f(\mathbf{r}_{t+k+1}, \mathbf{w}) \right\}. \tag{4.16}$$

As analysed by Perny and Weng (2010), this equation does not hold if f is the Chebyshev formula. To highlight this, consider the following situation described by Roijers et al. (2013). In an episode, the agent follows a policy π that selects two actions which corresponding rewards are $\mathbf{r}_1 = (0,3)$ and $\mathbf{r}_2 = (3,0)$. Hence, when the discount factor $\gamma = 1$, then $\mathbf{V}^\pi = (3,3)$. If we now apply the Chebyshev function with $\mathbf{z}^* = (3,3)$ and $\mathbf{w} = (0.5, 0.5)$ on the vector-valued value function, we get on the one hand

$$\begin{aligned} f(\mathbf{V}^\pi, \mathbf{w}) &= -\max(|0.5 \times (3-3)|, |0.5 \times (3-3)|) \\ &= -\max(|0.5 \times 0|, |0.5 \times 0|) \\ &= 0 \end{aligned} \tag{4.17}$$

While on the other hand, when applying f throughout the learning process, we see

$$\begin{aligned} E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k f(\mathbf{r}_{t+k+1},\mathbf{w})\right\} &= \gamma^0 f(\mathbf{r}_1,(0.5,0.5)) + \gamma^1 f(\mathbf{r}_2,(0.5,0.5)) \\ &= -\max(|0.5\times(0-3)|,|0.5\times(3-3)| \\ &\qquad -\max(|0.5\times(3-3)|,|0.5\times(0-3)|) \\ &= -\max(|0.5\times-3|,|0.5\times 0| - \max(|0.5\times 0|,|0.5\times-3|) \\ &= -\max(1.5,0) - \max(0,1.5) \\ &= -1.5-1.5 \\ &= -3 \\ &\neq f(\mathbf{V}^\pi,\mathbf{w}) \end{aligned} \tag{4.18}$$

This inequality has important consequences in reinforcement learning. As a result of the lack of additivity of the non-linear Chebyshev function, the Bellman equation no longer holds and the Chebyshev function in combination with a learning algorithm is not proven to converge to the optimal policy (Perny and Weng, 2010; Roijers et al., 2013). This is an important disadvantage of the Chebyshev scalarisation function since it would have been a powerful alternative to overcome the limitations of the linear scalarisation function as it could potentially discover every Pareto optimal solution. Nevertheless, it serves as an attractive research question to investigate the impact of this theoretical limitation on some practical exercises.

4.3.4 Experimental validation

To verify the empirical performance of the Chebyshev MORL algorithm, we focus on two environments with similar characteristics. These are the *Deep Sea Treasure* and the *Bountiful Sea Treasure* environments. In the following subsections, we will empirically analyse the effectiveness of the Chebyshev scalarised MORL algorithms in attaining the Pareto optimal solutions for specific configurations.

Deep Sea Treasure world

The Deep Sea Treasure (DST) is proposed by Vamplew et al. (2010) and is a standard MORL benchmark instance. The problem concerns a deterministic episodic task where an agent controls a submarine, searching for undersea treasures. The world consists of a 10×11 grid where 10 treasures are located, with larger values as the distance from the starting location increases. A visualisation of the environment is depicted in Figure 4.8. At each time step, the agent can move into one of the cardinal directions, i.e., up, down, left and right.

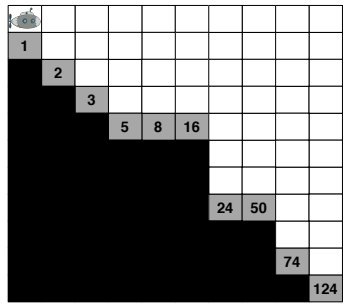


Figure 4.8: A visual representation of the Deep Sea Treasure world

The goal of the agent is to minimise the time needed to reach the treasure, while maximising the treasure value.[3] In this environment, a Pareto optimal policy is a path to a treasure that minimises the Manhattan distance to each of the 10 treasures. Given the reward signal of the environment, the Pareto front is entirely non-convex and presented in Figure 4.9.

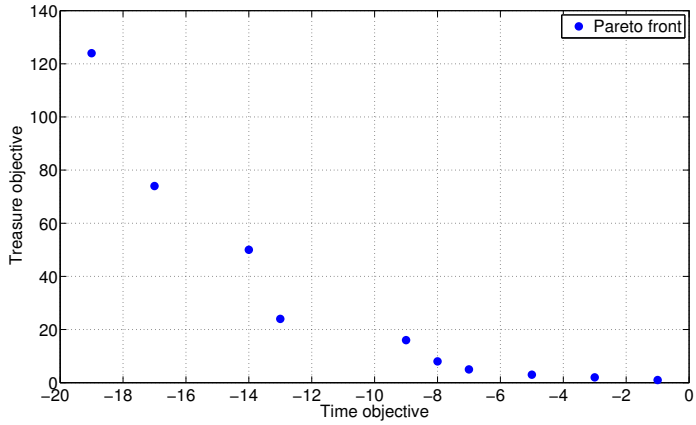


Figure 4.9: The Pareto front of the Deep Sea Treasure world

To thoroughly analyse the performance of the Chebyshev scalarised MORL algorithm, we investigate the converged policy after 10.000 episodes for different parameter configurations. For each configuration, we specify a weight vector $\mathbf{w} \in [0, 1]$ and a static attraction point $\mathbf{z}^* \in \mathbb{R}^m$. We evaluate which policy the algorithm converges to for 11 weight vector and 30×140 attraction points. Therefore, in total we evaluate 46200 possible configurations. In Figure 4.10, we depict on the x and y-axis the first and the second

[3] Traditionally, single-objective reinforcement learning solves a maximisation problem. If the problem at hand concerns a minimisation of one of the objectives, negative rewards are used for that objective to transform it also into a maximisation problem.

coordinate of the $\mathbf{z}^*$ vector, respectively. The z-axis represents the corresponding weight used for the time objective, i.e., w_1, while the second objective, i.e., the treasure objective, is $w_2 = (1 - w_1)$. Eventually, the colour of the dot represents which of the 10 Pareto optimal policy has been retrieved when following a greedy policy at the end of the learning phase. In case the retrieved policy was not Pareto optimal, no dot is displayed.

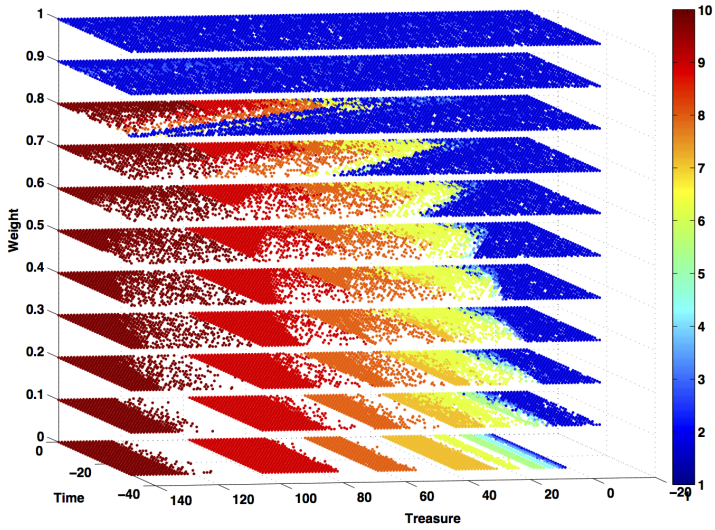


Figure 4.10: A parameter sweep for different weight vector and attraction points in the Deep Sea Treasure world

From the figure, we see different trends emerging. For starters, we see that for specific configurations every possible Pareto optimal policy has been learned as all the possible colours are present. To some degree, this result confirms the findings of the Chebyshev scalarisation function in general multi-objective optimisation also in MORL. Secondly, we see that the weight vector, usually employed by the decision maker to emphasise a specific trade-off between solutions, does not provide an explicit bias to specific parts of the objective space. For instance, when the weight for the time objective is 0, we find different Pareto optimal solutions. On the contrary, when the weight is equal to 1, almost regardless of the value of the attraction vector, the policy that retrieves the smallest treasure is learned. Thirdly, for different combinations of weight vector and attraction vector, we see several areas emerging where particular solutions are optimal. When these areas become more obvious also the space that separates these areas and where no Pareto optimal policy is discovered, becomes more distinctive. This can easily be noticed when comparing the areas for weights 0.5 and 1.

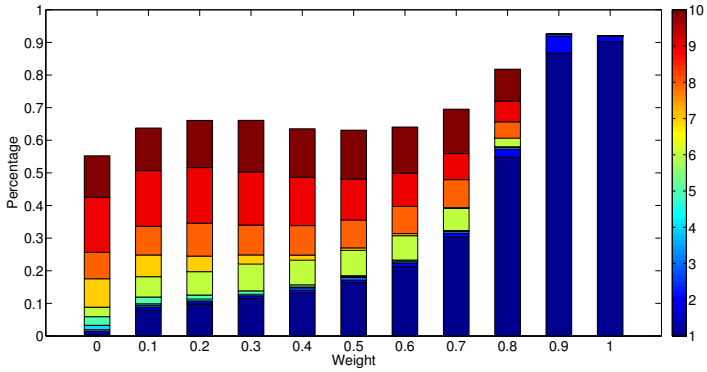


Figure 4.11: The retrievable policies for different weights in the Deep Sea Treasure world

In Figure 4.11, we approach this analysis from a different viewpoint. In this stacked bar chart, we investigate for each of the 11 weights the occurrence of each Pareto optimal policy. With this representation, we can straightforwardly analyse in what percentage of the cases the Chebyshev MORL algorithm does not converge to an optimal solution. We see that for weights w_1 from 0 to 0.6, the percentages of occurrence of each of the Pareto optimal solutions remain relatively identical. We note that for these weights also over 30% of the time no Pareto optimal solution was learned, which is remarkable. Only for the larger weights, we see a decrease in the amount of dominated policies but also in the diversity of the optimal solutions, i.e., for weight $w_1 = 1$ in 90% of the cases the optimal policy towards the smallest treasure is learned.

From the previous figures, one could argue that certain weights, such as $w_1 = 0.4$, offer a relatively good setting to obtain various Pareto optimal trade-off solutions, if one plays around with the values of the reference point. However, the robustness of these parameter settings is also crucial, i.e., with the same configuration, do we always retrieve the same solution of the policy space. In the next experiment, we repeat 11 runs of the Chebyshev algorithm while keeping $w_1 = 0.4$ fixed and varying the reference point $\mathbf{z}^*$. In Figure 4.12, we then denote the configurations that consistently yield the same Pareto optimal policy throughout the 11 runs. As we see, only for some very specific parameter configurations, the Chebyshev algorithm consistently learns the same policies. More specifically, this is the case in only 9.59% of the 4200 configurations examined. Therefore, this demonstration clearly shows the huge amount of variability of the Chebyshev single-policy MORL algorithm, which will become a constant factor in each of the subsequent experiments.

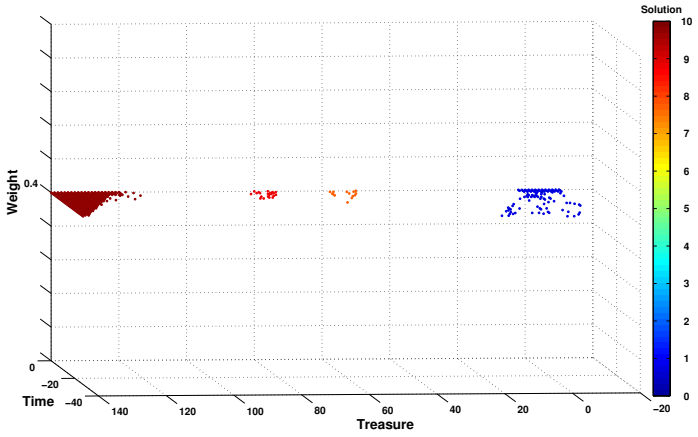


Figure 4.12: The variability of the Chebyshev single-policy MORL algorithm over multiple runs with $w_1 = 0.4$ is very large. In only 400 out of 4200 parameter configurations, the same Pareto optimal policy is returned. In the other configurations, the results varied and the algorithms converged to a different policy.

Bountiful Sea Treasure world

Before we proceed to formulating conclusions on the Chebyshev scalarised MORL algorithm, it is fruitful to investigate its effectiveness also on another environment with other characteristics. The Bountiful Sea Treasure environment is a variation to the Deep Sea Treasure environment where the reward signal is altered to accommodate for a convex Pareto front. A visual representation of the environment and its Pareto front with 10 elements is provided in Figure 4.13 and Figure 4.14, respectively.

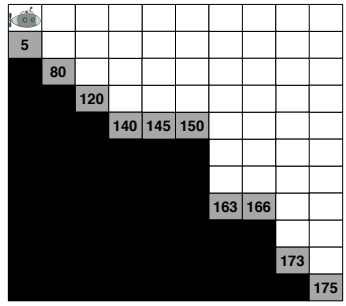


Figure 4.13: A visual representation of the Bountiful Sea Treasure world

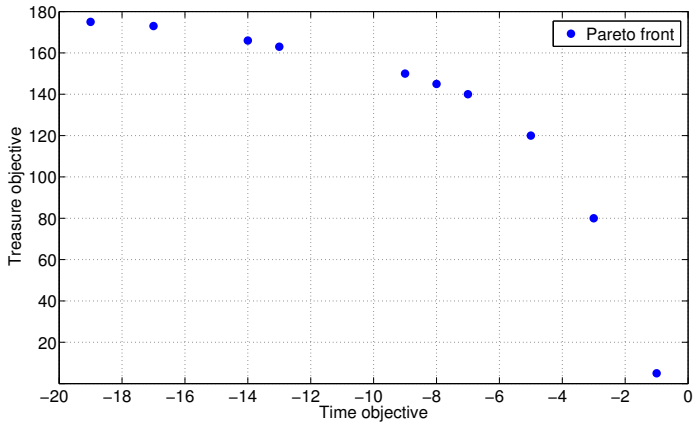


Figure 4.14: The Pareto front of the Bountiful Sea Treasure world

Since both environments only differ from one another in their reward signal, it is very suitable to run a similar parameter sweep. In Figure 4.15, we depict the learned policies for the same elements in the weight space and the space of attraction points. What strikes us immediately is the fact that the diversity of the policies has disappeared compared to the previous experiment. In a lot of cases, the extreme policies, denoted by red and blue dots, are learned almost regardless of the weight parameter and the attraction point. However, closely examining the results teaches us that still every Pareto optimal policy is being learned, albeit for some policies only for a very restricted set of configurations.

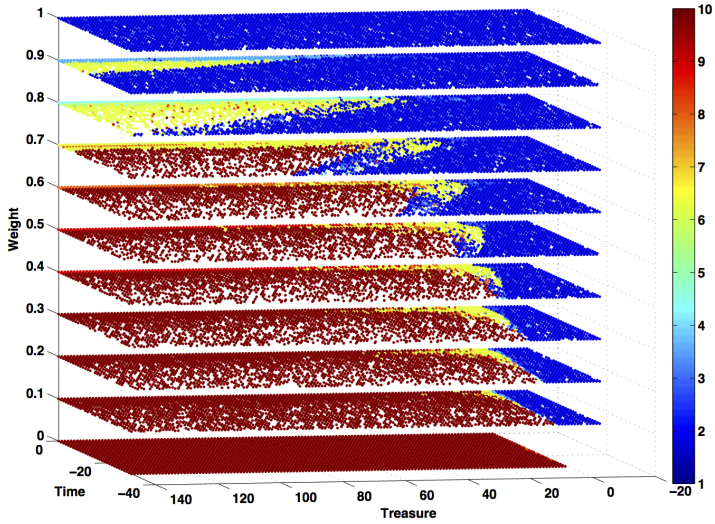


Figure 4.15: A parameter sweep for different weight vector and attraction points in the Bountiful Sea Treasure world

In Figure 4.16, we denote the probability of learning each Pareto optimal policy for different weight configurations. We note that for the extreme weights, i.e., $w_1 = 0$ or $w_1 = 1$, a single Pareto optimal policy is learned with a large probability. Only in about 7% of the cases, the Chebyshev scalarised MORL algorithm yielded a dominated solution, which is a lot less than in the previous experiment. For the other weights that entail a more realistic compromise, we see a different pattern emerging. As the value of the weight increases, we see that the probability of learning the policy to the smallest treasure decreases while the probability of the policy retrieving the largest treasure increases. Near the centre of the weight space, we note that the probability of learning a dominated policy is the largest.

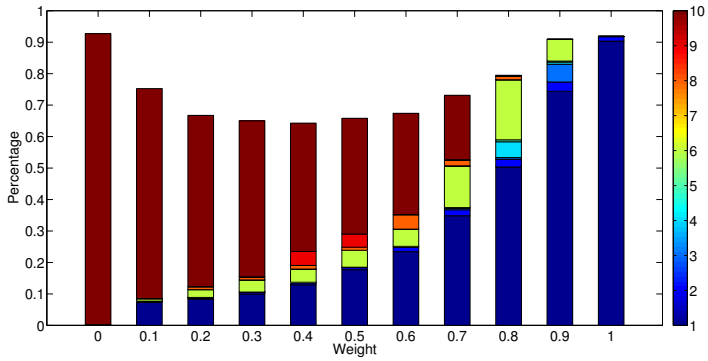


Figure 4.16: The retrievable policies for different weights in the Bountiful Sea Treasure world

In both environments, we have performed an in-depth analysis of the Chebyshev scalarised MORL algorithm for several parameter configurations. We can now present two statements on the efficiency and the effectiveness of the scalarisation function in combination with the MORL framework.

In terms of effectiveness, we have seen that in both environments every Pareto optimal policy was retrieved using a specific parameter configuration. However, especially in the convex environment, the frequency of some of the optimal solutions is very low, making it especially difficult to define a good weight vector and attraction point to retrieve a specific trade-off between objectives.

The efficiency of the Chebyshev scalarised MORL algorithm is relatively low, i.e., we have noted that in both environments in 10 to 30 percent of the situations, the algorithm did not succeed in finding a Pareto optimal solution. The reason for this behaviour can be found in the fact that the Chebyshev scalarisation function does not preserve the additivity relation. In this example, we see that this means that the scalarisation function is not guaranteed to converge to a Pareto optimal solution in every run of the algorithm. This is a serious limitation as the decision maker is not concerned in retrieving suboptimal solutions. In the case where the decision maker prefers robustness and consistency over the effectiveness of the solution, he might opt employing a MORL algorithm with a linear scalarisation function instead of the same algorithm with a non-linear Chebyshev scalarisation function. In the following section, we will more closely compare both scalarised MORL algorithm on several benchmark environments.

4.4 Benchmarking linear and Chebyshev scalarised MORL

To properly benchmark the two scalarised MORL techniques, it is necessary to obtain simulation environments where the set of optimal policies in known. Given this information,

we can then determine how close the learned policies approach the Pareto front of optimal policies. The environments used for benchmarking the performance of both scalarisation functions are the Deep Sea Treasure, Mountain Car and Resource Gathering environments. Before we proceed to the empirical evaluation, we present a distance metric that allows to compare the quality of the learned policies to their optimal values.

4.4.1 Performance criteria

In single-objective optimisation, the return of a policy can easily be compared to its optimal value since the rewards are scalar. A result of this sole dimensionality of the reward signal, there exists only a single optimal policy or multiple optimal policies but with an identical return. In the multi-objective case, the rewards are vectorial and usually not one, but multiple, optimal policies exist. Therefore, researchers have been investigating into quality indicators that analyse the obtained multi-objective solutions on multiple criteria as well. The two most important criteria are the degree of optimality and the spread. Moreover, the decision maker is interested in solutions that approach the Pareto front as closely as possible but also in solutions that are diverse in the objective space to offer a disperse set of trade-off solutions.

Many quality indicators exist, but the one that is the most interesting for our context is the hypervolume indicator (Zitzler et al., 2003a; Van Moffaert et al., 2013a). The hypervolume measure is a quality indicator that evaluates a particular set of vectorial solutions by calculating the volume with respect to its elements and a reference point (Figure 4.17). As the goal is to maximise the hypervolume, this reference point is usually defined by determining the lower limit of each objective in the environment. The hypervolume indicator is of particular interest in multi-objective optimisation as it is the only quality measure known to be strictly increasing with regard to Pareto dominance and it can be used for environments with an m-dimensional objective space.

4.4.2 Parameter configuration

We use the same parameter configuration for both scalarised MORL algorithms in order to investigate how closely their learned policies approach the Pareto front of the environment. Therefore, we need to retrieve a set of learned policies first. A simple and straightforward approach is to run the single-policy algorithm multiple times and to collect the resulting policies in a set. For instance, one can run the linear scalarisation algorithm many times for varying weights to approximate the Pareto front (Castelletti et al., 2002; Vamplew et al., 2010, 2008). In this experiment, we employ a naive but standard scheme to vary the weights which consists of a fixed step size of 0.1. In a bi-objective environment, the weights of the first objective w_1 are a consecutive sequence of $0, 0.1, 0.2, \ldots, 1.0$, while the corresponding weight of the second objective is $(1 - w_1)$ to satisfy Equation 3.22. Therefore, there are 11 weights in a bi-objective environment and 64 possible weights in an environment with 3 objectives. In Chapter 5, we investigate more complex schemes

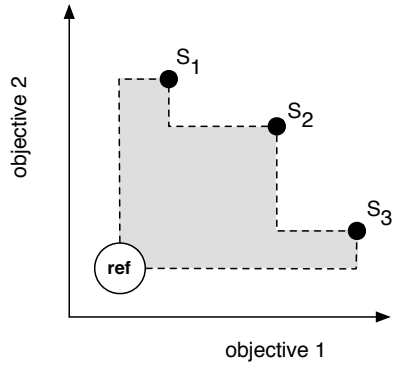


Figure 4.17: Illustration of the hypervolume calculator. It calculates the area of a set of non-dominated policies, i.e., S_1, S_2 and S_3, in the objective space with a given reference point *ref*.

that take into account the shape of the Pareto front the previously obtained solutions to determine the next weight.

For the Chebyshev scalarisation function, the attraction point is an additional parameter that needs to be specified. A common approach is to adjust the parameter during the learning process to continuously regulate the direction of search (Klamroth and Jørgen, 2007; Dächert et al., 2012). In practice, we redefine the attraction point every episode by analysing the highest return for each objective individually, plus a small positive constant. In our experiments, this constant was assigned to 1.

To accurately benchmark the performance of single-policy scalarised MORL algorithms, we propose a *train* and *test* setting for each iteration. In the train setting, we learn for a single episode with a specific weight parameter and an ϵ-greedy action selection strategy with $\epsilon > 0$. In the test phase, we analyse the policy the agent currently has converged to by determining the result of a greedy policy ($\epsilon = 0$). Each iteration, the return of this greedy policy for a particular weight is then stored in a set. We name this set the *approximation* set as it contains learned policies that aim to approximate the Pareto front of the environment. In earlier work, Vamplew et al. (2010) propose to employ the hypervolume indicator on this approximation set. The hypervolume of the approximation set can then be compared to the hypervolume of the *true* Pareto front, i.e., the set of Pareto optimal policies of the environment. The value of the reference point of the hypervolume metric is usually defined by determining the lower limit of each objective in the environment. Therefore, the value for the reference point of the hypervolume indicator is environment-specific and can be found in Table 4.1

Parameter	Value
Deep Sea Treasure	(-25, 0)
Mountain Car	(-350, -250, -500)
Resource Gathering	(-50, -10, -10)

Table 4.1: The value for the hypervolume indicator used to evaluate the policies for each of the environments.

The Q-values are initialised optimistically to encourage exploration. We average the results over 50 independent trials and depict the 95% confidence interval every 300 iterations. The other parameter settings such as the learning rate, the discount factor and the ϵ parameter can be found in the table below.

Parameter	Value
α	0.1
γ	0.9
ϵ	0.1

Table 4.2: The parameter settings for the linear and Chebyshev scalarised MORL algorithm.

4.4.3 Deep sea treasure world

We compare both scalarised MORL algorithms on a familiar environment, that is the non-convex Deep Sea Treasure world. In Figure 4.18, we depict the learning graphs for the linear and Chebyshev scalarised MORL algorithms. As the optimal policies in the environment form a non-convex Pareto front, the linear scalarisation function is struggling to approach the Pareto front. After around 100 iterations, the algorithms exclusively learns the two extreme policies of the Pareto front. Hence, a static hypervolume value with zero deviation is denoted in the figure. The Chebyshev scalarised MORL algorithm is able to improve the quality of its learned policies over time and surpasses the performance of its linear variant. However, in the end, not every Pareto optimal policy has been learned as its line does not coincide with the line of the Pareto front. This is merely a result of our naive and restricted set of weight parameters and reference point values as we have seen in Section 4.3.4.

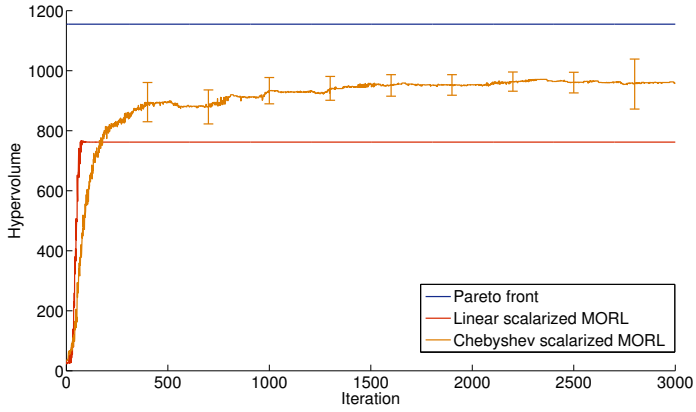


Figure 4.18: The linear and Chebyshev scalarised MORL algorithms on the Deep Sea Treasure world

These result are also emphasised in Figure 4.19, where we present the probability of converging to one of the Pareto optimal policies in the Deep Sea Treasure environment. As we see, the linear scalarised MORL does not succeed in learning policies other than the extremes of the Pareto front, which is a serious limitation. As a positive side note, we can verify that there was no occurrence where this scalarisation function did not converge to a dominated policy. The Chebyshev scalarised MORL algorithm also learned these extreme policies with a very large probability, but also other policies are retrieved, be it with a very small probability. The policy to treasure value 24 is the only one that was not discovered by the Chebyshev scalarised MORL algorithm. This is the case because that particular treasure is located at only one timestep from the treasure with value 50. The difference in time cost needed to reach the latter treasure is only 1, where the advantage in terms of treasure value compared to the former is $(50 - 24)$. Therefore, the step size of the corresponding weight interval, i.e., steps of size 0.1 in $[0, 1]$, did not allow to assign significant credit to the treasure with value of 24. In a few cases, also a dominated policy was learned.

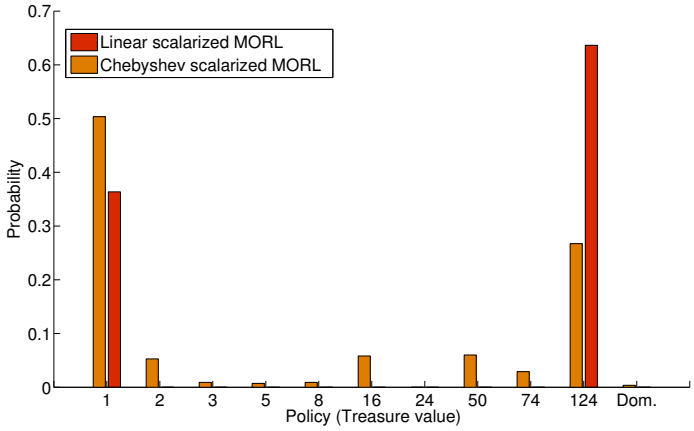


Figure 4.19: The retrieved policies for the linear and Chebyshev scalarised MORL algorithms on the Deep Sea Treasure world

4.4.4 Mountain car world

The single-objective Mountain Car world is a famous benchmark for reinforcement learning algorithms. In this world, a car is required to escape a one-dimensional valley. As the car's engine is less powerful than gravity, the vehicle must perform a series of acceleration and reversal actions to build enough potential energy to escape the valley. A visual representation of the environment can be found in Figure 4.20. The state space of the environment is a continuous domain of velocity and position coordinates. Traditionally, one would use function approximation techniques to deal with the continuous state space, such as for instance tile coding (Sutton and Barto, 1998). To simplify the setting, we discretised both continuous domains into a 6×6 grid. The action space consists of three actions: full throttle forward, full throttle backward, and zero throttle. The multi-objective version of this benchmark is challenging as it consists of three objectives that are to be optimised. The three objectives are (1) the time required to escape the valley and (2) the number of acceleration and (3) reversal actions, which are all to be minimised. The Pareto optimal set contains 470 dominating policies and the maximum amount of steps allowed to reach the goal location is 500 steps (Vamplew et al., 2010). It is important to note that the shape of the Pareto optimal set has a significant portion of locally convex and non-convex areas.

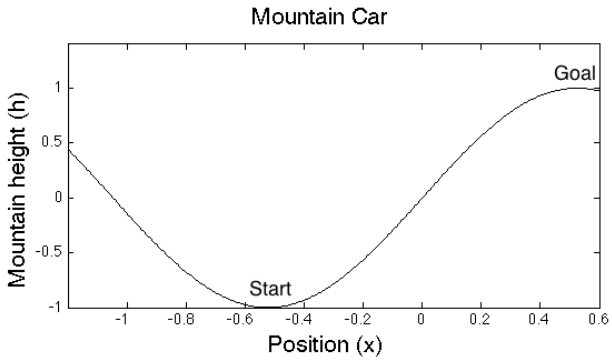


Figure 4.20: A visual representation of the Mountain Car environment.

The learning graph for both scalarisation function is presented in Figure 4.21. The Chebyshev scalarised MORL algorithm is learning faster and smoother than its linear variant. The learning curve of the former smooths out after a few hundred iterations, while the line of the latter is still fluctuating a lot. The line of the Pareto front is, however, still at significant distance. This is logical as we tried to approach the Pareto front containing 470 policies with only 64 different weight parameters.

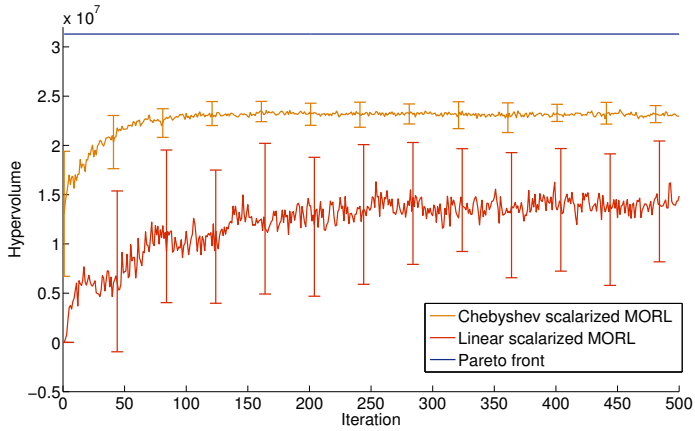


Figure 4.21: The linear and Chebyshev scalarised MORL algorithms on the Mountain Car environment.

As the Pareto front is very large, it is interesting to see where the learned policies are situated in the objective space. Therefore, we construct a 3D plot that represents the Pareto non-dominated solutions found by the two learning algorithms and the Pareto optimal set in Fig. 4.22 (a). Closer investigation of the policies teaches us that the linear

scalarised MORL algorithm retrieved only 25 distinct Pareto non-dominated policies, which are not necessarily members of the Pareto front. On the contrary, the Chebyshev scalarised MORL algorithm obtained 38 distinct non-dominated solutions. Note that both algorithms were run only 64 times, i.e., using 64 weight tuples. We also observe that the policies obtained by the linear scalarisation method are, as expected, located in convex areas of the Pareto optimal set, while the Chebyshev function learned policies that are situated in both convex and non-convex regions.

Next to optimality, the goal of a MORL algorithm is to ensure a significant spread in the solution space. Ideally, when the obtained results are scattered across the objective space, the decision maker has a larger choice amongst very different policies. On the contrary, when policies are clustered into particular areas, often the differences between those policies are minimal and they do not provide the decision maker with a diverse set of choices. In Figure 4.22 (b), (c) and (d), we present a sliced representation of the 3D plot where we focus on only two objectives at a time. Note that the Chebyshev method obtains a larger spread in the objective space, where the results of the linear scalarisation function are clustered into particular regions.

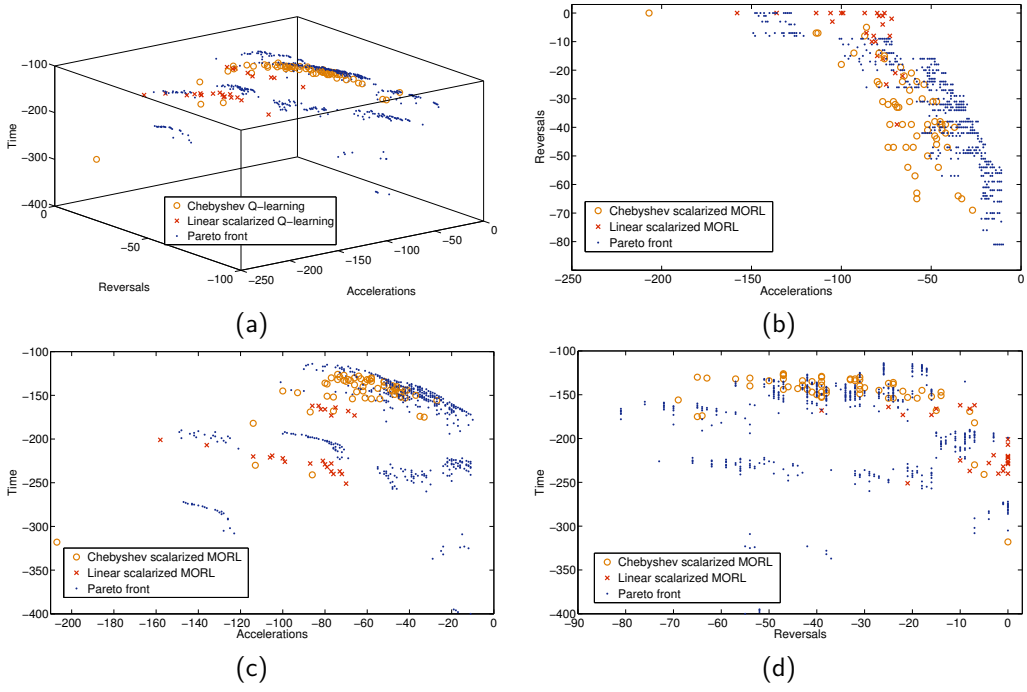


Figure 4.22: The policies in the objective space of the linear and Chebyshev scalarised MORL algorithms on the Mountain Car environment.

4.4.5 Resource gathering world

In this final experiment, we analyse the performance on an entirely convex environment, named the Resource gathering task. In this environment, adapted from Barrett and Narayanan (2008), the agent goes on a quest to bring gold and gems to its home site. Upon returning to its starting location, the goods he acquired are analysed. The possible gathering outcomes are bringing home nothing, gold, gems or gold and gems. A visual representation of the environment is provided in Figure 4.23. The three objectives are time, gold, and gems, and there are 4 Pareto optimal policies in this environment, i.e.,

1. a policy that returns to the home site as fast as possible without carrying anything,
2. a policy that gathers the gold and returns as fast as possible,
3. a similar one for the gems objective,
4. and a policy that gathers both resources as fast as possible.

These optimal policies are all located on the convex hull of the Pareto front as depicted in Figure 4.24.

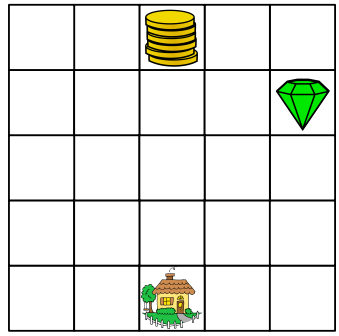


Figure 4.23: A visual representation of the Resource Gathering environment.

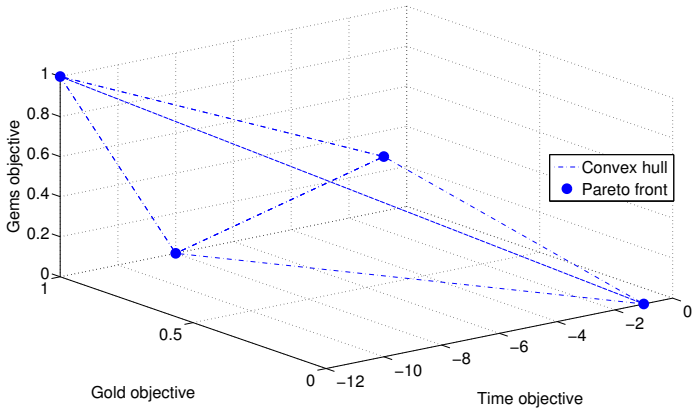


Figure 4.24: The Pareto front of the Resource Gathering environment.

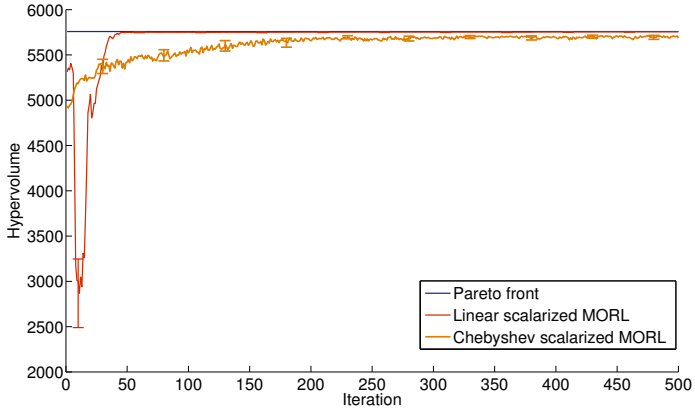


Figure 4.25: The linear and Chebyshev scalarised MORL algorithms on the Resource Gathering environment.

The result of learning is presented in Figure 4.25. We see that the linear scalarised MORL learns very fast and retrieves the entire set of Pareto optimal policies after roughly 50 iterations. This behaviour was to be as expected as the Pareto front is entirely convex. Only in the initial learning phases, a dominated policy was learned, but in future iterations every Pareto optimal policy was retrieved. As presented in Figure 4.26, we note that, also in this environment, the probability of learning one of the Pareto optimal policies is not uniform. As we will see in the following chapter, this is entirely the effect of the weight parameters.

From the learning graph, one might have the impression that the Chebyshev scalarised MORL algorithm is once again performing relatively well as its line approaches the line of

the Pareto front. However, additional results have shown that increasing the number of iterations does not help in learning the entire Pareto front. The bar chart in Figure 4.26, also teaches us that in many cases, the simplest policy, i.e., the one that returns to the home location as fast as possible without any artefacts, is learned while the policy that retrieves both gems and gold is never learned. Also, in more than 50% of the cases, a dominated policy was learned with the Chebyshev scalarised MORL algorithm, which is not a good indication.

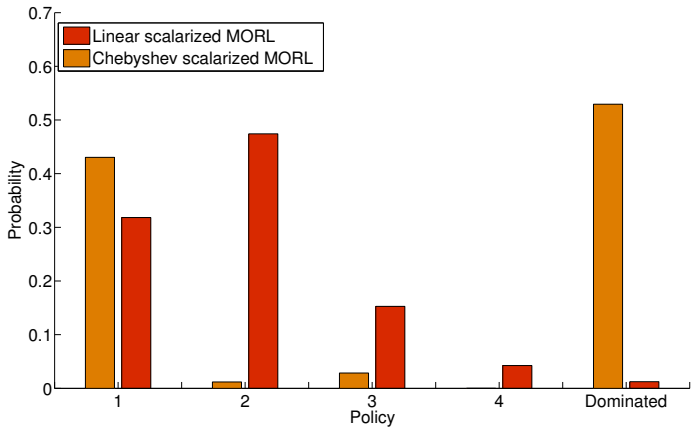


Figure 4.26: The retrieved policies for the linear and Chebyshev scalarised MORL algorithms on the Resource Gathering environment.

4.5 Conclusions

In this chapter, we have investigated single-policy MORL that employ scalarisation functions. In this branch of MORL, the decision maker wants to obtain a single optimal policy that satisfies its interest, based on a weight vector that is priorly defined. In order to accommodate for a multidimensional feedback signal, we have proposed a general framework for scalarised MORL. This framework learns and updates $\mathbf{Q}$-vectors, which comprise a standard Q-value for each individual objective. In the action selection strategy, these $\mathbf{Q}$-vectors are scalarised given a certain scalarisation function.

We have argued two instantiations for scalarisation functions, but the linear scalarisation function is the most common approach. We have investigated the theoretical properties of the linear scalarisation function and we have seen they also hold in the case of a reinforcement learning setting. More precisely, the linear scalarised MORL algorithm is guaranteed to converge to a Pareto optimal solution, which is an essential property for reinforcement learning in general. As a downside, not every Pareto optimal policy is discoverable but only the policies that are situated on the convex hull can be retrieved.

In general multi-objective optimization, the Chebyshev scalarisation function is proven to be able to discover every Pareto optimal solution, regardless of the shape of the Pareto front. However, researchers have analysed that this property does not hold in MORL. In this chapter, we empirically investigate to what degree this theoretical limitation effects the behaviour of the learning agent in practice. After thoroughly investigating both scalarisation functions on three benchmark instances, we have seen that in most cases, the Chebyshev scalarised MORL performs acceptably and is able to find policies the linear scalarised MORL algorithm does not manage to retrieve. As a drawback, this performance is very parameter-specific and in a significant amount of cases, the Chebyshev scalarised MORL indeed converges to a suboptimal policy. As a conclusion we could state that the Chebyshev scalarised MORL algorithm does not solve all the limitations of the linear scalarisation function but it has the potential to significantly improve its performance in terms of the optimality of the solution and their spread in the objective space. In its strictest form, one could state that non-linearity and MORL inherently exclude each other due the lack of additivity. However, the search for non-linear scalarisation functions that can be used to enhance the performance of single-policy optimisation should not take a halt at this point. We highlight three scenarios where non-linear scalarisation function can still be employed and that can potentially guide future research.

For starters, one could combine temporal difference learning with a non-linear scalarisation function in pure form, that is without any refinement or restrictions. In such case, the scalarisation function should be used as a heuristics that is not founded on any theoretical principles but that can guide the search to potential fruitful areas of the objective space. The work in Van Moffaert et al. (2013b,a,c) falls in this category.

Secondly, it is possible to combine temporal difference learning with a non-linear scalarisation function, be it in a more restricted form. To solve the issue of non-additivity, one can alter either the learning algorithm itself or its state representation (Geibel, 2006). Nevertheless, this approach remains very specific and complex. Additionally, as mentioned by Roijers et al. (2013), the speed of learning suffers from this workaround. Yet, it would undoubtedly be of great interest to further investigate this path.

Thirdly, one could investigate the use of non-linear scalarisation functions also outside the temporal difference setting. Instead of adopting a value-iteration approach where a value function is learned, one could learn a policy directly. These are policy-search methods which directly and iteratively update a single policy without learning a value function first. In that facet, *policy-gradient* methods for MORL have also been proposed. In Uchibe and Doya (2007), a policy-gradient algorithm is proposed that learns in a direct and iterative manner a single policy satisfying predefined constraints. This algorithm is called *constrained policy gradient reinforcement learning* (CPGRL) as it iteratively updates the policy in the direction of a gradient satisfying one or more constraints. As CPGRL does not learn in a temporal difference manner, it poses no problem in combining it with a non-linear scalarisation function.

4.6 Summary

In this chapter, we investigated single-policy MORL that rely on scalarisation functions to transform the dimensionality of the reward signal to a single dimension. We have proposed a framework that allows to reuse standard reinforcement learning algorithms in a multi-objective setting by employing a scalarisation function in the action selection strategy. We have theoretically and empirically analysed the linear and the non-linear Chebyshev scalarisation function and emphasised their advantages and limitations. We have seen that non-linearity and temporal difference learning do not synthesise well due to the lack of additivity of the scalarised returns. However, we have illuminated several paths for future research that can boost the development of learning algorithms that handle a non-linear scalarisation function.

5 | Iteratively learning multiple multi-objective policies

In the previous chapter, we have investigated into MORL algorithms that employ a scalarisation function and a weight vector to discover a single trade-off solution. In this chapter, we explore how this single-policy setting can be augmented to obtain multiple policies by adaptively varying the weight vector. The main question to research is then how to consecutively redefine the weight vector in other to find a diverse set of trade-off solutions taking into account the relation between the weight vector and the slope of the Pareto curve. As pointed out in Das and Dennis (1997), this relation is non-isomorphic, meaning that the mapping from the weight space to the converged solution in the objective space is unclear. In this chapter, we will dig deep into this problem and highlight the reasons for this relation. We will propose two tree-based mechanisms that allow to define adaptive weights in the case of a discrete and continuous Pareto front. These techniques are experimentally validated on several benchmark problems and conclusions are formalised.

In more detail, we will present the following contributions in this chapter:

- Analyse how single-policy algorithms can obtain multiple policies
- Investigate the non-isomorphic mapping from weight to objective space
- Propose an adaptive weight algorithm that can efficiently explore discrete Pareto fronts
- A similar set of algorithms that can efficiently explore continuous Pareto fronts

This research has been published in Van Moffaert et al. (2014a) and in Van Moffaert et al. (2013).

5.1 Iteratively learning an archive of trade-off solutions

Over the years, a few mechanisms have been proposed to obtain a set of trade-off solutions with single-policy algorithms. Usually, a single-policy algorithm is run multiple times and its resulting policies are collected in an *archive* or approximation set, denoted by $\mathcal{A}$. For instance, one can run the linear scalarisation algorithm many times for varying weight vectors to approximate the convex hull of the Pareto front (Castelletti et al., 2002; Vamplew et al., 2008, 2010). In the case the Pareto front is continuous, it is both impossible and unnecessary to retrieve every Pareto optimal solution. In this respect, the idea is to find an archive that is both (i) (close to) optimal and (ii) diverse, using a predefined number of weights. Diversity is a very important aspect as it is important to guarantee that the policies are not clustered into particular areas of the objective space, but that they identify diverse trade-offs that the decision maker can choose from. In Figure 5.1 (a), we depict a poor archive since some policies tend to be clustered in specific areas while leaving other regions of the objective space uncharted. In Figure 5.1 (b), the policies are more widespread and, therefore they define a better approximation.

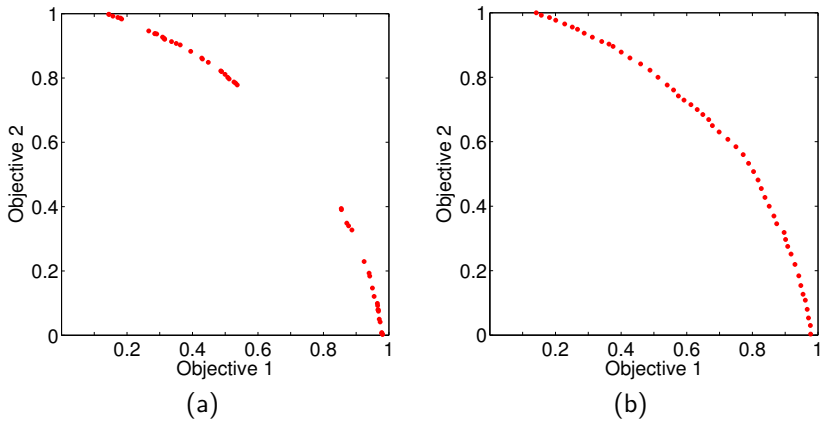


Figure 5.1: A poor approximation of a continuous Pareto front is shown in (a), while in (b) the policies are more diverse and spread across the objective space.

A common strategy is to uniformly and evenly divide the weight space in chunks of equal size and to collect the results. Nevertheless the approach, being simple and intuitive, is proven not to provide an even spread of solutions in the objective space. Moreover, as pointed out by Das and Dennis (1997), only for very specific shapes of the Pareto front, an even spread in the weight space provides an even spread in the objective space. Basically, without any prior knowledge on the shape of the Pareto front, it is impossible to foresee what point in the objective space a certain weight vector will yield. In layman's terms, learning with a weight vector of $(0.8, 0.2)$ does not guarantee that the solution obtained will compromise 80% of the best solution for the first objective and 20% of the best

solution of the second objective. This idea is illustrated in Figure 5.2 (a) and (b) which uses the same visualisation as in Section 4.2.2. In both figures, we depict the solutions A, B and C that correspond to three weight vectors, represented by the angles θ, θ' and θ'', respectively. [1] Although the three angles roughly divide the weight space in three parts of equal size, we see that the retrieved solutions do not uniformly cover the Pareto front in the objective space. In Figure 5.2 (b), the influence of the shape of the Pareto front is even larger since the solutions are now more clustered and at uneven distance from each other than in Figure 5.2 (a). These examples highlight that it is of crucial importance to

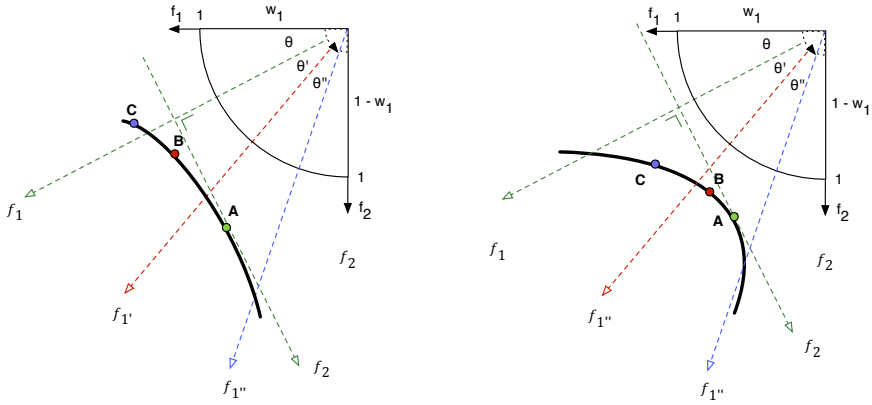


Figure 5.2: Depending on the shape of the Pareto front in (a) and (b), different solutions A, B and C are found for three identical weight vectors, defined by the angles θ, θ' and θ''.

investigate heuristics that consecutively define and redefine the weights vectors. In this regard, in Section 5.3, we describe several heuristics that implicitly extract the shape of the Pareto front and that guide the search in future iterations.

In the case that the Pareto front is discrete and only contains a limited number of solutions, adaptive weight heuristics can be employed to efficiently explore the objective space to find each and every optimal solution. In this discrete context, spread is not an important aspect but the retrieval of all solutions is. Therefore, we investigate how to explore the multi-dimensional objective space as fast and as efficient as possible by proposing an adaptive weight algorithm that uses a tree-based search mechanism. This algorithm will be analysed in Section 5.2.

[1] For clarity reasons, the line f_2' and f_2'' perpendicular to f_1' and f_1'' are omitted in the illustration

5.1.1 Related Work

Several researchers have attempted to define adaptive weighting schemes in the case of general optimisation problems. For continuous Pareto fronts, Dubois-Lacoste et al. (2011) proposed a series of algorithms that alter the weight of a linear scalarisation function with a so-called *dichotomic* scheme to specify the direction of search. As the optimisation algorithm relies on a linear scalarisation of objectives, only solutions on the convex hull of the Pareto front are retrieved, as we have seen in Section 4.2. These adaptive weight algorithms are created specifically for stochastic local search algorithms. Internally, they combine interesting techniques such as divide-and-conquer by either progressively exploring finer levels or by moving the search towards solutions that comprise the largest gap according to several metrics.

In Das and Dennis (1998), the *normal boundary intersection* (NBI) algorithm is proposed. NBI is able to produce an evenly distributed set of solutions given an evenly distributed set of parameters. In its internal workings, NBI breaks the original problem down into a set of subproblems, each with a specific scalarisation scheme. This scalarisation scheme is indifferent of the relative scales of objective functions but it is not guaranteed to retrieve optimal solutions in non-convex regions of the Pareto front.

To overcome the limitation of the linear scalarisation function concerning non-convexity, Kim and de Weck (2005) suggest to combine an adaptive weighting scheme with additional inequality constraints. Their algorithm divides the original problem into several subproblems that each focus on particular regions of the objective space. By continuously refining these regions, the algorithm is able to discover an even spread of solutions in both convex and non-convex regions of the Pareto front.

These approaches have proven their usefulness in general optimisation processes involving state-less and usually deterministic problems. However, they are not ideal for sequential multi-objective reinforcement learning problems where the state space can be large and where both the transition function as the reward signal can be stochastic. In Section 5.2 and 5.3, we propose two mechanisms that allow to repeatedly use single-policy MORL algorithms to obtain multiple policies in both discrete as well as continuous Pareto fronts.

5.2 Adaptive weight algorithms for discrete Pareto fronts

In some situations the system engineer might have some insights in the solution space of the problem. For instance, in case the dynamics of in the environments are rather deterministic and the number of actions limited, the number of optimal solutions in the objective space usually restricted. However, for the system engineer to find the appropriate weight configurations that retrieve all solutions with minimal computational effort is not trivial. To stress this statement, one can investigate the correspondence of the weight space to the objective space of the Bountiful Sea Treasure environment, previously introduced in Section 4.3.4. The objective space of the environment contains 10 Pareto optimal solutions that are each retrievable with a certain weight vector. In Figure 5.3 (a), we

denote an exhaustive sweep of the weight space and the converged policy in the objective space. On the x-axis, we depict the weight w_1 assigned to the first objective while the weight of the second objective is $w_2 = 1 - w_1$. We see that several areas in the weight space emerge that each converge to the same solutions. However, it is clear that some policies correspond to a much larger set of weights than others, e.g., the blue policy occurs much more frequently than the yellow policy. Additionally, some policies correspond to such a restricted set of weights that (1) a uniform sampling of the weight space will be biased towards particular policies and (2) it is unrealistic that the system engineer would ever find it by purely guessing the weights. This aspect is highlighted in Figure 5.3 (b) where we depict a zoomed-in portion of a particular area of Figure 5.3 (a). We note that the interval of weights that converge to the green policy is very narrow and located between the yellow and turquoise policies. Subsequently, when the number of objectives increases, also the number of decision variables, i.e., the weights, increases, which makes the problem of retrieving all optimal solutions even harder.

Therefore, we elaborate on an algorithm that efficiently explores the weight space of any m-dimensional problem in order to find every optimal solution with as minimal computational effort as possible. This algorithm will be unveiled in the upcoming section.

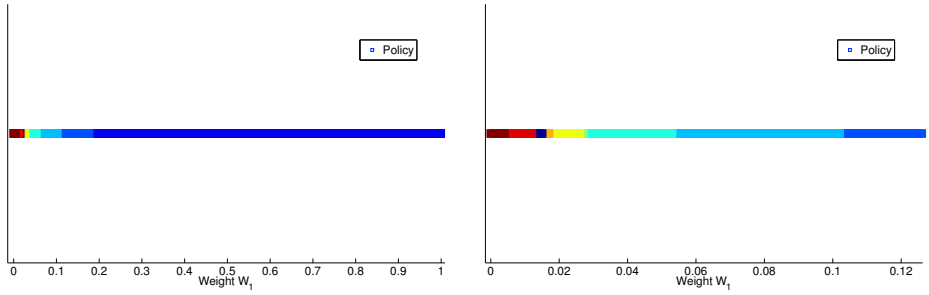


Figure 5.3: In (a), we depict the mapping of weight space to objective space of the Bountiful Sea Treasure environment. In (b), we present a zoomed-in area of (a) and we note that some policies are only retrievable for a very restricted set of weights.

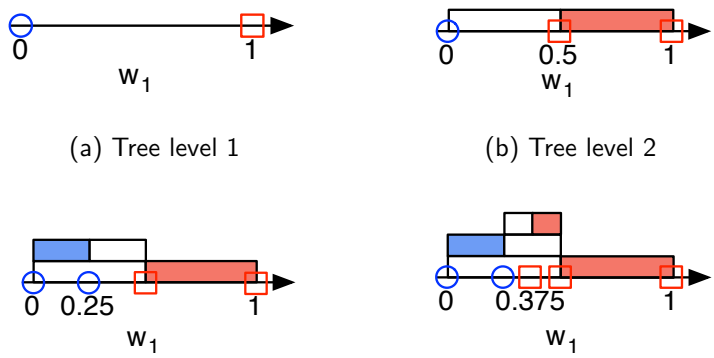
5.2.1 The algorithm

The algorithm we propose is a tree search algorithm that can be incorporated with an underlying single-policy MORL algorithm, such as the linear scalarised MORL algorithm of Section 4.2. The tree search algorithm incrementally supplies a sequence of weight configurations to efficiently explore the weight space. In computer science, searching is often performed with tree-like structures as their hierarchical composition makes them very efficient for searching purposes.

Our tree search algorithm exploits two properties of convex linear combinations. First, we know that the weight space for linear scalarisation is always $(m-1)-$dimensional, with m the number of objectives. This is due to the fact that all weights must sum to 1, and therefore setting all but one weight is enough to also fix the value for the last weight. Selecting a specific weight assignment $\mathbf{w} = (w_1, w_2, \ldots, w_m)$ transforms the MOMDP into a standard MDP where the optimal policy is also the Pareto optimal policy of the original MOMDP.

Another important aspect of convex Pareto optimal sets is that the set of policies that is optimal for a weight $w_o \in [0, 1]$ are precisely those policies whose line-representations lie on the upper convex envelope of the Q-functions (Ehrgott, 2005), i.e., if for two different weight assignments $\mathbf{w}$ and $\mathbf{w}'$ the same policy is optimal, it is also optimal for the weight assignments between $\mathbf{w}$ and $\mathbf{w}'$.

The aim of our method is to take benefit from these properties and exploit the structure of convex hulls to identify the weight space regions that are associated with a particular Pareto optimal policy, and specifically by recursively refining the boundaries of these regions.



(a) Tree level 1

(b) Tree level 2

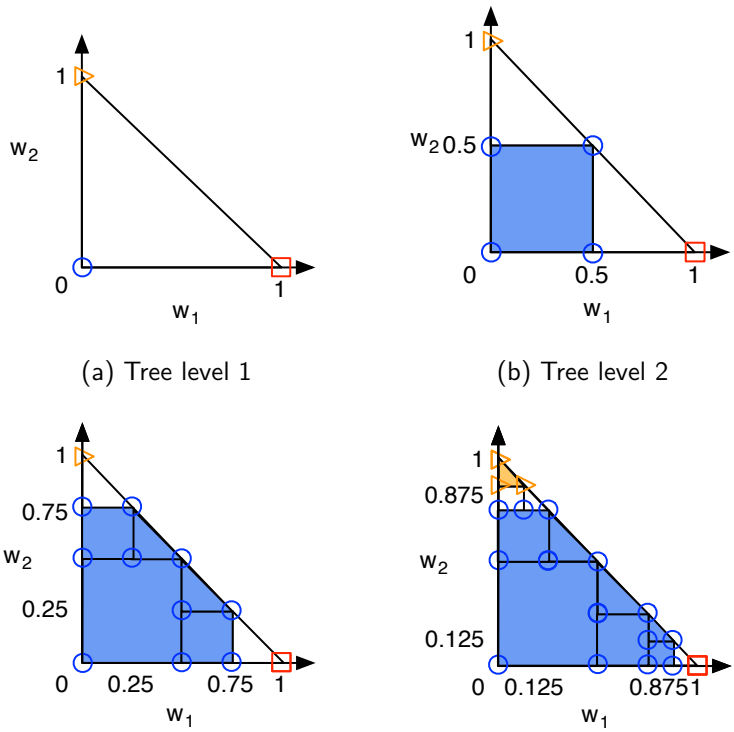
(c) Tree level 3

(d) Tree level 4

Figure 5.4: A visualisation of the exploration of the weight space of a bi-objective problem. In this case, the weight w_1 (x-axis) is the only decision variable, since $w_2 = 1 - w_1$. Different symbols and colours identify different Pareto optimal policies.

Initially, the algorithm starts by solving the MOMDP using the linear scalarised MORL algorithm of Chapter 4 with the weight combinations that define the boundaries of the entire weight space. For instance, for a bi-objective problem, the two extrema are $(w_1 = 0, w_2 = 1)$ and $(w_1 = 1, w_2 = 0)$. As a running example, we visualise the search process for a bi-objective toy problem in Figure 5.4 (a), with different symbols and colours identifying different Pareto optimal policies. If the discounted cumulative rewards of the converged policies are identical, our search stops since we can conclude that there is only one Pareto

optimal policy within the corresponding interval in the weight space. Note that different policies can be optimal for a given weight assignment, but their discounted cumulative rewards must be the same. If the discounted cumulative rewards for the policies found are different, as is the case in our toy problem, the region bounded by these weight assignments is split into equal parts. In our example, the interval is split in the middle of the original weight configurations. Subsequently, the scalarised MORL algorithm is run again with the weight values at the split point, as can be seen in Figure 5.4 (b). Thus, we build a binary tree corresponding to the regions in the weight space. Using breadth-first search, we efficiently search the tree to identify the weight space regions associated with different policies, by only expanding nodes, i.e., splitting regions, if some of the boundary points yield different policies. After four iterations on our toy problem, two Pareto optimal policies are identified, with the intervals currently estimated at $[0, a]$ and $[a, 1]$ respectively, with a somewhere between 0.25 and 0.375 . Further refinements of the boundary may reveal new intervals lying in $]0.25, 0.375[$ that yield other Pareto-optimal policies.



(a) Tree level 1

(b) Tree level 2

(c) Tree level 3

(d) Tree level 4

Figure 5.5: A visualisation of exploration of the weight space of a problem involving three objectives. In this case, the weight w_1 (x-axis) and w_2 (y-axis) are the decision variables, since $w_3 = 1 - w_1 - w_2$. Different symbols and colours identify different Pareto optimal policies.

Logically, the tree structure and its splitting strategy are dependent on the number of objectives. In the case of only two objectives, the tree is a binary tree and the splitting mechanism only operates on a single dimension since evaluating a single new point is sufficient. In environments with three and four objectives, the tree structure becomes a quadtree and octree, respectively. Also, the number of split points increases with the number of objectives. In the event of three objectives, the number of split points is 5, i.e., one point in the middle of each edge of the square, and one in the middle. This process is visualised in Figure 5.5. Note that the upper-right part of the weight space is infeasible, since in that region $w_1 + w_2 > 1$.

In general, we look for center points in lower dimensional figures. Since an environment with m objectives consist of $(m-1)$ dimensions that are free, since the weight of the last objective, w_m is entirely dependent on weights w_o with $o \in [0, m-1]$. Therefore, we are

iterating through geometrical hypercubes lower than dimension $(m-1)$, i.e., dimension 1 (points) to dimension $(m-2)$. Moreover, we know that a hypercube of dimension z contains 2^z vertices, e.g., a cube has 8 vertices. Each of these vertices of the z-dimensional figure plays a role in one or more lower dimensional figures of dimension $(z-1)$. For instance, a vertex of a cube is part of three faces of dimension 2. Therefore, we can use the combinatorial binomial coefficient to know how many times a vertex in a space with $(m-1)$ dimensions can be used in a space with only $k < (m-1)$ dimensions. This is $\binom{m-1}{k}$. Of course, we are counting some doubles, so we have to subtract 2^k. In the formula below, we calculate the sum of lower dimensional figures by iterating with k. Additionally, we also add one point, this is the center point of the $(m-1)$ dimensional figure itself.

$$1 + \sum_{k=1}^{m-2} 2^{m-1-k} \binom{m-1}{k} = 1 + \sum_{k=1}^{m-2} 2^{m-1-k} \frac{(m-1)!}{k!(m-1-k)!}. \tag{5.1}$$

For example, for an environment with four objectives, the tree structure is an octree and the geometrical figures we are splitting are cubes. In that case, the number of split points to be evaluated is equal to the number of edges, i.e., one split point in the middle of each edge, plus the number of faces, i.e., one split point in the middle of each face, plus 1 split point for the middle of the cube:

$$1 + 12 + 6 = 19. \tag{5.2}$$

5.2.2 Experimental evaluation

In this section, we will perform an empirical evaluation of different weight schemes on several MORL benchmark instances.

5.2.3 The Bountiful Sea Treasure world

We first analyse the tree-based algorithm on the standard Bountiful Sea Treasure (BST) environment of Section 4.3.4. The correspondence of the weight space to the objective space of that environment is already depicted in Figure 5.3. In the experiments, we compare our algorithm to two alternative weight schemes that are often employed in literature. The first scheme is a predefined sequence of weights that divide the weight space in chunks of equal size. The second scheme is an unbiased, uniform random weight selection strategy. We analyse the results by investigating the cardinality, i.e., the number of optimal elements obtained over time compared to the size of the Pareto front.

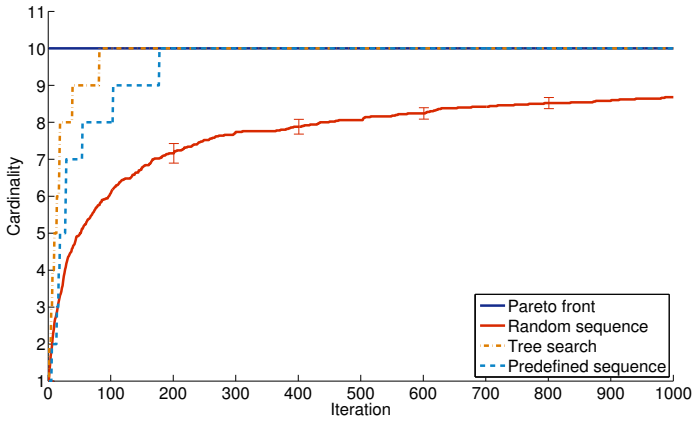


Figure 5.6: Number of Pareto optimal policies identified as a function of the number of weight combinations tried on the Bountiful Sea Treasure environment.

In Figure 5.6, we depict a cumulative sum of the cardinality of the optimal policies for the random and the predefined weight sequence and the tree search algorithm. All of the results are averaged over 50 runs and the 95% confidence interval is depicted at regular time intervals. Note that our deterministic tree search method efficiently explores the weight space and is able to obtain every element of the Pareto optimal set in under 100 iterations, where one iteration equals solving the MDP with a single weight combination. The predefined sequence also performs acceptably but it takes around 200 iterations before the entire Pareto front is retrieved. The random weight sequence performs worse as it only finds 8 Pareto optimal policies after 1000 iterations. The fact that the random and the predefined sequence are still able to obtain 8 policies is due to the still somewhat reasonable sizes of most of the weight space regions leading to different policies. In the three-objective version of this world in Section 5.2.5, we will see that the difference with the random method becomes much larger, as the size of most regions shrinks dramatically.

5.2.4 The Resource Gathering world.

The Resource Gathering world of Section 4.4.5 contains three objectives. Therefore, the weight space in Figure 5.7 is two-dimensional, as only w_1 and w_2 need to be set as $w_3 = 1 - w_1 - w_2$.

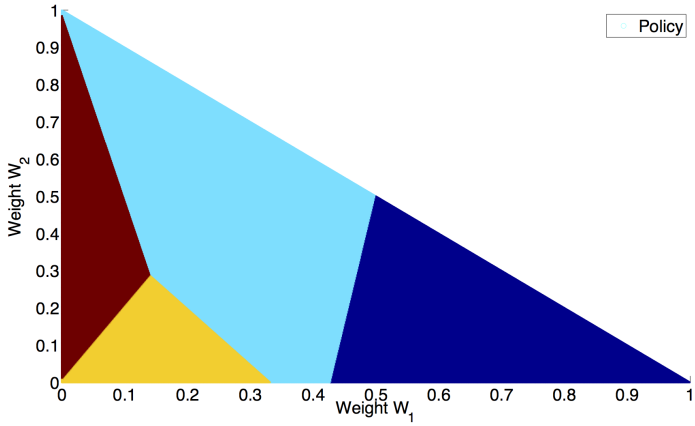


Figure 5.7: The four Pareto optimal policies in the Resource Gathering world and its associated locations in the weight space.

From this figure, we can clearly see the four large, distinct regions, which should ease the discovery of the four Pareto optimal policies. In Figure 5.8 we summarise the performance on this benchmark instance. We notice that, although there are no huge differences in size between the various regions, the random sequence on average needs up to 50 iterations. The predefined weight sequence is doing a bit better although still 22 weight configurations are tried before every solution is found. We also see that the performance of this scheme is often stagnating and plateaus are formed where a lot of weights are often retrieving policies that were found in previous iterations. Obviously, this is because the scheme provides weights that are at a predefined distance from one another rather than determined adaptively. Finally, the tree-based search algorithm only requires 5 iterations at minimum cost of resources.

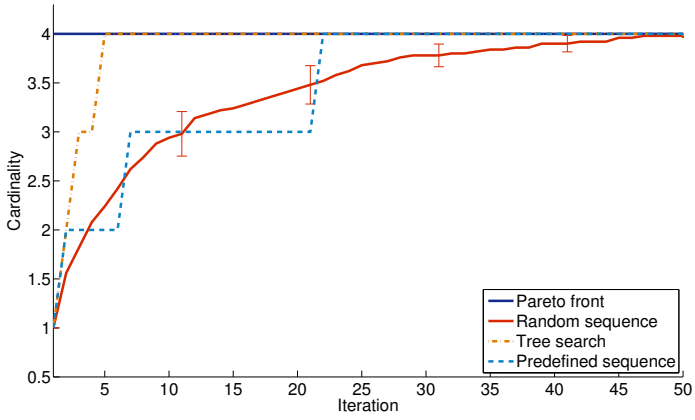
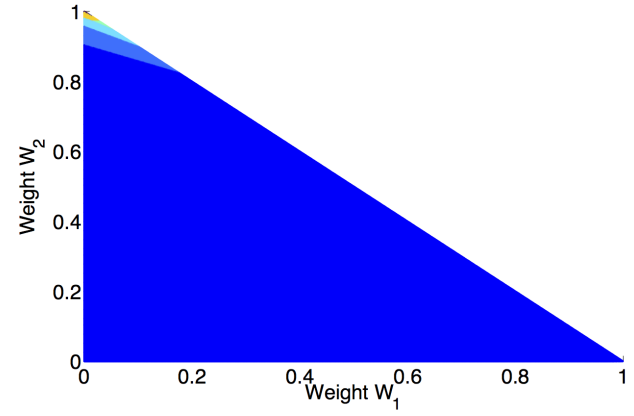


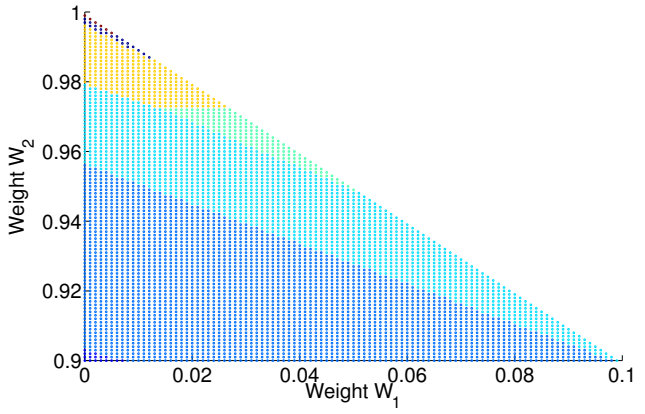
Figure 5.8: The number of Pareto optimal policies identified as a function of the number of weight combinations tried for different search algorithms on the Resource Gathering world.

5.2.5 The Bountiful Sea Treasure 3D world

Additionally, we extend the two-dimensional BST by adding a third objective, namely the pressure objective. This objective relates to the depth of the submarine and a penalty is given linear to this depth, i.e., at every step, the agent receives a reward of $-$depth. As a result, the Pareto optimal set still consists of 10 optimal policies, but instead of allowing for any shortest path to a treasure, the submarine must travel as long as possible in the shallowest water before making its descent to a particular treasure. Therefore, the Bountiful Sea Treasure 3D world is the most challenging environment of the three discussed here. As is clear from the weight space in Figure 5.9 (a), a single policy is optimal in 95% of the space, while the other optimal policies are only obtained for a very specific set of weights (see the zoomed-in representation in Figure 5.9 (b)). Even using a brute-force exploration of the weight space, while trying all weight combinations with increments of 0.001, only allowed to discover 7 of the 10 Pareto optimal policies. This again shows that uniformly sampling the weight space to identify the convex parts of the Pareto optimal set can be a very inefficient procedure indeed.



(a)



(b)

Figure 5.9: (a) The weight space of the Bountiful Sea Treasure 3D with its associated Pareto optimal policies. A large part of the weight space converges to the same blue policies while other policies correspond to a very restricted collection of weights. (b)The zoomed-in portion of the top-left of (a). Only 7 out of 10 Pareto optimal policies are identified using an exhaustive brute-force search.

In Figure 5.10, we show the search performance of the different weight schemes. It is clear that a random sequence only finds a limited subset of the Pareto front at a very slow pace. The predefined sequence performs very well in the first 1500 iterations of the experiment but stagnates afterwards. The boost in early iterations is credited to the vertical order in which the weight space is explored in our implementation. If the weight space is to be explored horizontally, it would vary w_1 first instead of w_2 and then the blue policy would be constantly retrieved for the first 10000 iterations. Finally, our tree search is able

to identify all Pareto optimal policies, although requiring many more evaluations than on the previous worlds. This is due to the extremely small size of the weight space regions associated with some of the Pareto optimal policies. Note that there is some stochasticity in the search process, since the learning agent did not converge at every iteration, resulting in a misclassification of the weight combination. Nevertheless, our algorithm is able to deal with this stochasticity and eventually retrieves all Pareto optimal solutions.

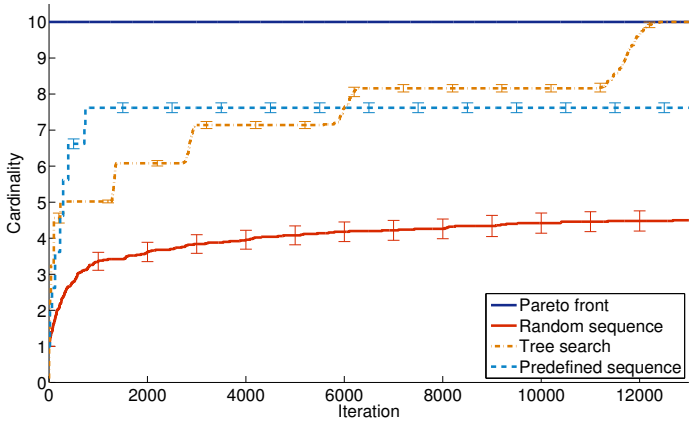


Figure 5.10: The number of Pareto optimal policies identified as a function of the number of weight combinations tried for different search algorithms on the Bountiful Sea Treasure 3D world.

Figure 5.11 shows the resulting quad-tree of the tree search method. Colours indicate the depth of the tree, with red being the deepest levels. The figure clearly shows that the algorithm is able to concentrate on the regions containing several policies, while avoiding those other regions that correspond to the same policy. Figures 5.12 and 5.13 also confirms this, by showing which weight combinations are evaluated by the tree search algorithm. The algorithm intensively explores the boundaries between different regions, and ignores the interiors of these regions, as they will not yield new policies. Refining the boundaries between regions can help find very small regions that are associated with other Pareto optimal policies.

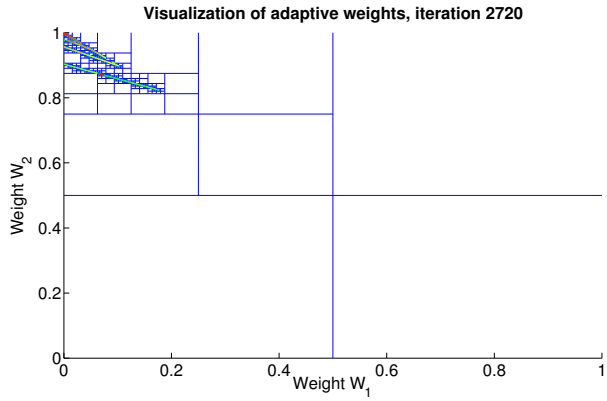


Figure 5.11: The resulting quad-tree after 2720 iterations. The colours indicate the depth of the tree, red being the deepest levels. We see that the algorithm spends as little time as necessary on the portion of the weight space corresponding to the blue policy in Figure 5.9 (a) before concentrating on the small regions containing several policies

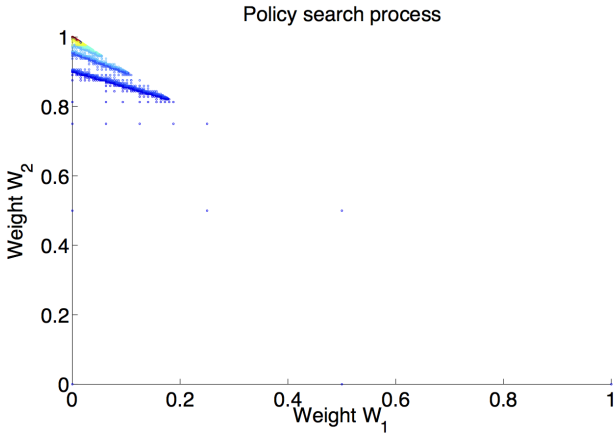


Figure 5.12: The search process of the algorithm, where each point is a weight that is evaluated. We note that the algorithm quickly intensifies its search towards the top-left of the space.

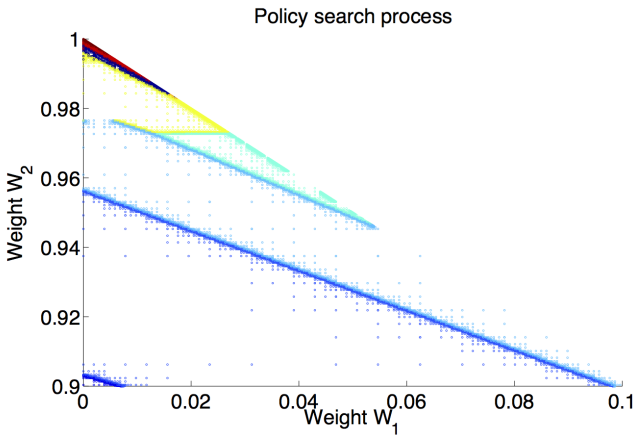


Figure 5.13: A zoomed-in portion of Figure 5.12. We clearly see that exploration is focussed on the boundaries between regions.

5.2.6 Advantages and limitations

We showed that a uniform sampling of the weight space often does not lead to a uniform sampling of the convex parts of the Pareto optimal set. Therefore, the decision maker should not only rely on intuition when tuning weights by hand, since it wastes significant amounts of computation power. To overcome these problems, the decision maker should be assisted with adaptive weight algorithms. In this regard, our algorithm performs breadth-first search on a tree-based partitioning of the weight space, and explores those regions that contain the boundaries between already identified policies. Even though the number of split points of the tree structure increases exponentially in the number of objectives, it is still preferable over non-adaptive schemes since sampling high-dimensional spaces is notoriously more complex. We experimentally showed it to outperform a uniform random sampling of the weight space, and that it can automate the process of tuning weights to find policies that achieve a desired trade-off between the different objectives. In some occasions a predefined weight sequence might show acceptable results but this performance is not guaranteed as the order in which the sequence followed is crucial to avoid plateau performance.

The tree-search algorithm serves as a simple and useful mechanism to learn multiple policies by repeatedly executing single-policy optimisation algorithms with different weights. Because the search strategy and the optimisation algorithm are inherently two distinctive components, the principle can also be combined with other optimisation techniques than MORL, such as for instance local search. The strategy is extendible to environments with a high number of objectives as only the structure of the tree changes. Another alternative to the process described here, is to use k-d trees, instead of quadtrees, octrees, etc. Since

k-d trees do not split regions symmetrically, but based on an inspection of the bounding weight assignments and the neighbouring regions, the search process could be sped up.

In case the environment is roughly deterministic and contains a discrete amount of optimal policies, the tree search algorithm will perform well. However, if a lot of stochasticity is present, the equality operator on which the search strategy is based to accentuate or ignore particular areas should be relaxed. In that case, a distance metric could be employed. If the distance between the policies then does not exceed a certain threshold, the policies are said to be roughly equal and the strategy should not emphasise their corresponding area in the weight space.

5.3 Adaptive weight algorithms for continuous Pareto fronts

In the previous section, we assumed the Pareto front contains a discrete amount of policies. However, in real-life applications, this assumption might no longer hold and the Pareto front could be continuous. If we would naively apply the tree-based strategy in this context, the algorithm would perform splits at every single level since the equality operator is too strict. Therefore, continuous Pareto fronts require the use of heuristics to yield an archive of solutions that is both optimal and well-spread. In the subsequent section, we will describe in more detail what these properties entail.

5.3.1 Defining good approximation set

Usually, a good archive is considered to contain solutions of high quality and that are well-spread to offer a diverse set of trade-offs. To formally specify these properties, one needs distance metrics to measure the space between the archive and the true Pareto front, if available. Currently, there is no single best measure available to evaluate the quality of an archive, but researchers often use quality indicators to asses its performance (Knowles and Corne, 2002; Zitzler et al., 2003b). Quality indicators usually transform a set of m-dimensional vectors into a real number:

> **Definition 5.1**
>
> A (unary) quality indicator is a function $I : \mathcal{A} \rightarrow \mathbb{R}$, which assigns each vector $(A_1, A_2, \ldots, A_p)$ of a set of m-dimensional elements to a real value $I(A_1, \ldots, A_p)$

So far in this dissertation, we have already dealt with one specific quality indicator. Remember the hypervolume measure of Section 4.4.1, which is also a quality indicator as it takes a set of vectors as input and provides a real number representing the volume of that set in the objective space. Other typical quality indicators are:

- Generational distance : A measure that calculates how far the elements in the archive, generated by the optimisation algorithm, are from those in the Pareto front (Van Veldhuizen and Lamont, 1998). Internally, the average Euclidean distance $dist$ from the elements of the archive $\mathcal{A}$ to the Pareto front PF is calculated. The lower the GD value, the closer the obtained elements are to the Pareto front:

$$GD(\mathcal{A}, PF) = \frac{1}{|\mathcal{A}|}\Big(\sum_{i=1}^{|\mathcal{A}|} dist(A_i, PF)^2\Big). \tag{5.3}$$

- Inverted generational distance : This variant of the generational distance indicator calculates the distance from the viewpoint of the Pareto front (Van Veldhuizen and Lamont, 1998). In essence, $IGD(\mathcal{A}, PF) = GD(PF, \mathcal{A})$:

$$IGD(PF, \mathcal{A}) = \frac{1}{|PF|}\Big(\sum_{i=1}^{|PF|} dist(PF_i, \mathcal{A})^2\Big). \tag{5.4}$$

- Generalised spread : A measure of diversity that calculates the extent of spread amongst the obtained solutions (Aimin et al., 2006). It calculates the ratio of the Euclidean distance from the extremes of the archive to the Pareto front to the mean distance between each point and its nearest neighbour. This way, one gets an idea of the density of the solutions in the archive.

Often not a single indicator is used in the performance assessment process but rather a combination of multiple quality indicators before conclusions are drawn. To be theoretically correct, Zitzler et al. (2003b) state that an infinite number of unary set measures is needed to detect in general whether one approximation is better than another one.

5.3.2 Existing adaptive weight algorithms

In Dubois-Lacoste et al. (2011), a series of adaptive weight algorithms (AWAs) is proposed to steer the search process. These algorithms are analysed in combination with *two-phase* local search (TPLS) optimisers. TPLS is a powerful algorithmic framework that comprises two phases and focusses exclusively on bi-objective problems. In a first phase, a single-objective local search algorithm obtains a high-quality solution for one of the objectives, while in the second phase this solution serves as a starting point for a sequence of future weight configurations. In the end, the goal of TPLS is to find an archive of high quality yet diverse trade-off solutions. Some TPLS variants use a predefined sequence, while others select the new weight as a function of the solutions previously discovered, and their coverage of the Pareto front. We summarise these existing variants in the following paragraph.

- TPLS: The standard procedure defines a sequence for w_1 ranging from 0 to 1 with steps of $\frac{1}{N_{scalar}}$, where N_{scalar} determines the step size. The weight for the other

objective, i.e., w_2 is calculated by $w_2 = 1 - w_1$ for the bi-objective case. However, although the method performs a uniform sampling of the weight space, there is no guarantee that the resulting solutions will yield a uniform spread in the objective space as we have seen earlier in this chapter. Another crucial aspect of TPLS is the sequential order of weights. When TPLS is stopped prematurely, it will not have sampled the latter part of the weight space, potentially leaving a part of the Pareto front uncharted. Hence, TPLS does not produce solutions as good as possible as fast as possible, i.e., it has poor *anytime* behaviour.

- RA-TPLS: To overcome these problems, the *Regular Anytime* TPLS algorithm (RA-TPLS) is proposed (Dubois-Lacoste et al., 2011). The algorithm explores the weight space in a divide-and-conquer manner by progressively exploring finer levels k. RA-TPLS starts at evaluating $w_1 = 0$ and $w_1 = 1$ at level 0. At the next level, when $k = 1$, $w_1 \in \{0.5\}$. At level $k = 2$, $w_1 \in \{0.25, 0.75\}$ and so forth. As the search continues, the coverage of the weight space is refined. Therefore, at any time, the search effort is (almost) evenly distributed across the possible weight settings.
- AN-TPLS: The previous two methods generate weights in a predefined manner. However, sometimes the shape of the Pareto front is irregular and the search direction should be adapted by taking into account the actual shape of the Pareto front. *Adaptive Normal* (AN) TPLS defines a metric to identify the largest gap in the coverage of the Pareto front (Dubois-Lacoste et al., 2011). Between all the currently obtained trade-offs, the pair with the largest gap according to the metric specified (Euclidean distance in this variant) is used to calculate the next weight, aiming to fill this largest gap. The new weight w_1 is perpendicular to the line between the objective function $\mathcal{F}$ of solutions s_1 and s_2 defining the largest gap in the objective space, assuming s_1 and s_2 are normalised (Dubois-Lacoste et al., 2011):

$$w_1 = \frac{\mathcal{F}_2(s_1) - \mathcal{F}_2(s_2)}{\mathcal{F}_2(s_1) - \mathcal{F}_2(s_2) + \mathcal{F}_1(s_2) - \mathcal{F}_1(s_1)} \tag{5.5}$$

- AN-TPLS-HV: An extension to the standard adaptive TPLS algorithm of above uses an alternative metric to specify a distance measure. The hypervolume measure is employed to measure the size of the gap in the Pareto front (Dubois-Lacoste et al., 2011). Given two solutions s_1 and s_2, the hypervolume measure calculates the rectangle defined in the objective space:

$$HV(s_1, s_2) = |(\mathcal{F}_1(s_1) - \mathcal{F}_1(s_2)) \cdot (\mathcal{F}_2(s_1) - \mathcal{F}_2(s_2))| \tag{5.6}$$

Although these methods are developed with local search algorithms in mind, they can be adapted to interact with other optimisation algorithms, such as reinforcement learning. In order to assess their performance, we proceed to an experimental evaluation of these existing techniques, before we propose some alternatives of our own.

5.3.3 Experimental evaluation

Before we proceed to the results, we depict the properties of the existing AWAs in a simulation environment, mimicking a distributed smart camera network. This environment is named CamSim. Afterwards, in Section 5.3.4, we propose novel extensions that overcome most of the limitations of the current AWAs.

Simulation environment

CamSim (Esterle et al., 2013) simulates a distributed smart camera network. [2] Smart cameras are fully computationally capable devices endowed with a visual sensor, and typically run computer vision algorithms to analyse captured images. Where standard cameras can only provide plain images and videos, smart cameras can pre-process these videos and provide users with aggregated data and logical information, such as the presence or not of an object of interest. Since smart cameras are designed to have a low energy footprint, their processing capabilities are also low. Communication between cameras allows the network as a whole to track objects in a distributed fashion, handing over object tracking responsibilities from camera to camera as objects move through the environment. In one approach (Esterle et al., 2014), cameras exchange object tracking responsibilities through auctions, sending auction invitations to other cameras, who may then bid to buy objects. The cameras use pheromone-based on-line learning to determine which other cameras they trade with most often. This neighbourhood relationship graph (the vision graph), enables them to selectively target their auction invitations and achieve higher levels of efficiency. In Lewis et al. (2013), six different behavioural strategies are available to cameras, which determine the level of marketing activity they undertake, given the learnt vision graph. Some strategies incurred higher levels of communication overhead but typically obtained higher levels of tracking confidence; other strategies obtained the opposite results. However, the trade-off realised by each strategy is found to be highly scenario dependent; as camera positions varied and object movements differed, the relative benefits of the strategies is greatly influenced.

Although cameras make decisions based on local information, we are primarily interested in performance at the global level. This consists of two network-level objectives:

1. Tracking confidence: the achieved tracking confidence during a small time window for each object by the camera tracking that object, summed over all objects. This objective is to be maximised.

2. Number of auction invitations: the number of invitations sent by all cameras as a result of auction initiations, during a small time window, a proxy for communication and processing overhead. This objective is to be minimised.

The camera agents are single-state independent learners and can choose between six marketing strategies defining each agents behaviour. The scalarised reward r_{total} is a

[2] The CamSim software is open-source and retrievable at https://github.com/EPiCS/CamSim

camera-specific reward and is a weighted-sum of the *utility* reward $r_{utility}$ and the negative *auction invitation* reward $r_{auction}$, given w_1 for the first objective and $(1 - w_1)$ for the second objective:

$$r_{total} = w_1 \times r_{utility} + (1 - w_1) \times -r_{auction}. \tag{5.7}$$

The utility reward of a camera i is calculated by

$$r_{utility} = \sum_{j \in O_i} [c_j \cdot v_j \cdot \phi_i(j)] - p + sp. \tag{5.8}$$

Here, v_j is a visibility parameter which is determined by the distance and angle of the observed object to the observing camera. The tracking performance is estimated by a confidence value c_j. Both values c_j and v_j are between 0 and 1 as soon as the observed object is within the field of view of a camera, 0 otherwise. In addition to utility earned by tracking objects, a camera b may make a payment to another camera s in order to 'buy' the right to track an object from that camera. This requires that the 'selling' camera s already itself owns the object. If an exchange is agreed, then the object is removed from the inventory of camera s and added to camera b. Moreover, p denotes the sum of all payments made in trades in that iteration, and sp conversely denotes the sum of all payments received (Esterle et al., 2011).

The $r_{auction}$ reward denotes the number of auction invitations sent by this camera at the current time step. Our aim is to minimise the number of auction invitations, but traditionally, RL concerns a maximisation problem. Therefore, we want to maximise the negative number of auction invitations in Equation 5.7. The agents use a softmax action selection strategy with τ equal to 0.2. For more information on the details behind these marketing strategies, we refer to Lewis et al. (2013).

Simulated data In our simulation, a scenario comprises a set of cameras with associated positions and orientations, along with a set of objects and their movement paths through the environment. We simulate and evaluate configurations within 11 qualitatively different scenarios using the open source CamSim software. We also acquired video feed data from a real smart camera network, which gives us a twelfth scenario. All simulated scenarios are depicted in Figure 5.14, where a dot represents a camera and the associated triangle represents its field of view.

Real-life data Figure 5.15 shows snapshots from each camera at five different points in time. Each camera captured 1780 frames, looped four times to create a total of 7120 frames, each with a resolution of 640×480.

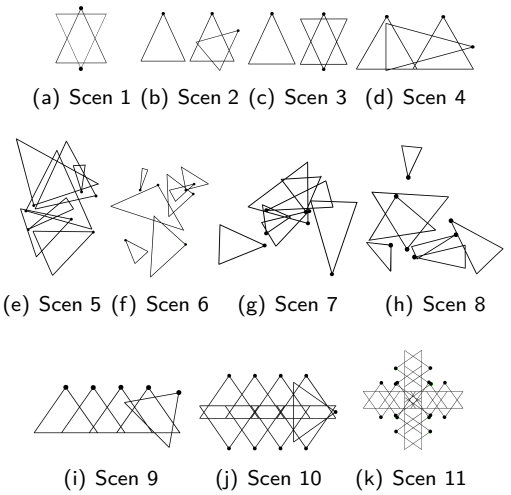


Figure 5.14: The scenarios tested with the CamSim simulation tool. A dot represents a camera, the associated triangle represents its field of view.

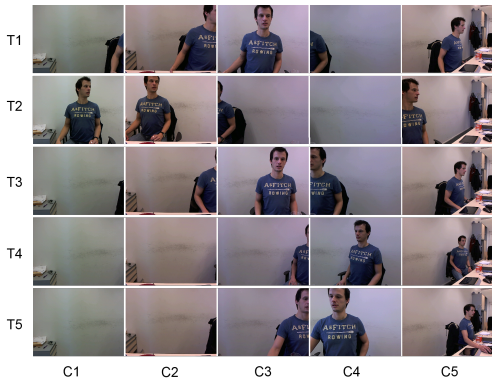


Figure 5.15: Shots from five participating cameras tracking a single person.

Results I

We will now present the results of applying the adaptive two-phase weight schemes in combination with reinforcement learning agents in CamSim. There are two main side marks. First, note that we do not use local search techniques as in the original TPLS proposals (Dubois-Lacoste et al., 2011) so therefore we refer to these implementations as two-phase reinforcement learning or TPRL. Secondly, in the current setup, the weight parameter used in each iteration of the simulation is the same for all agents in the scenario.

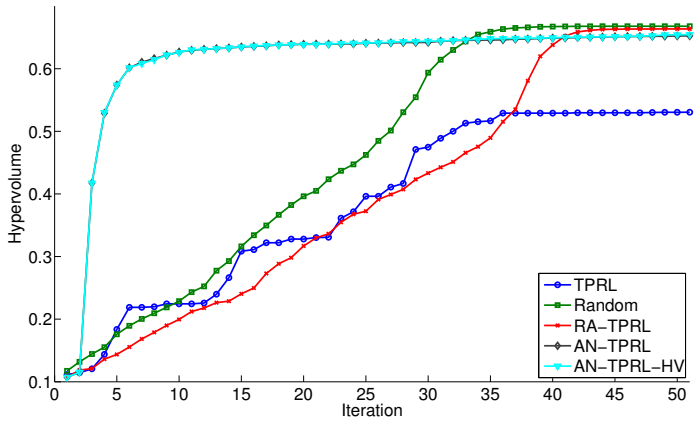
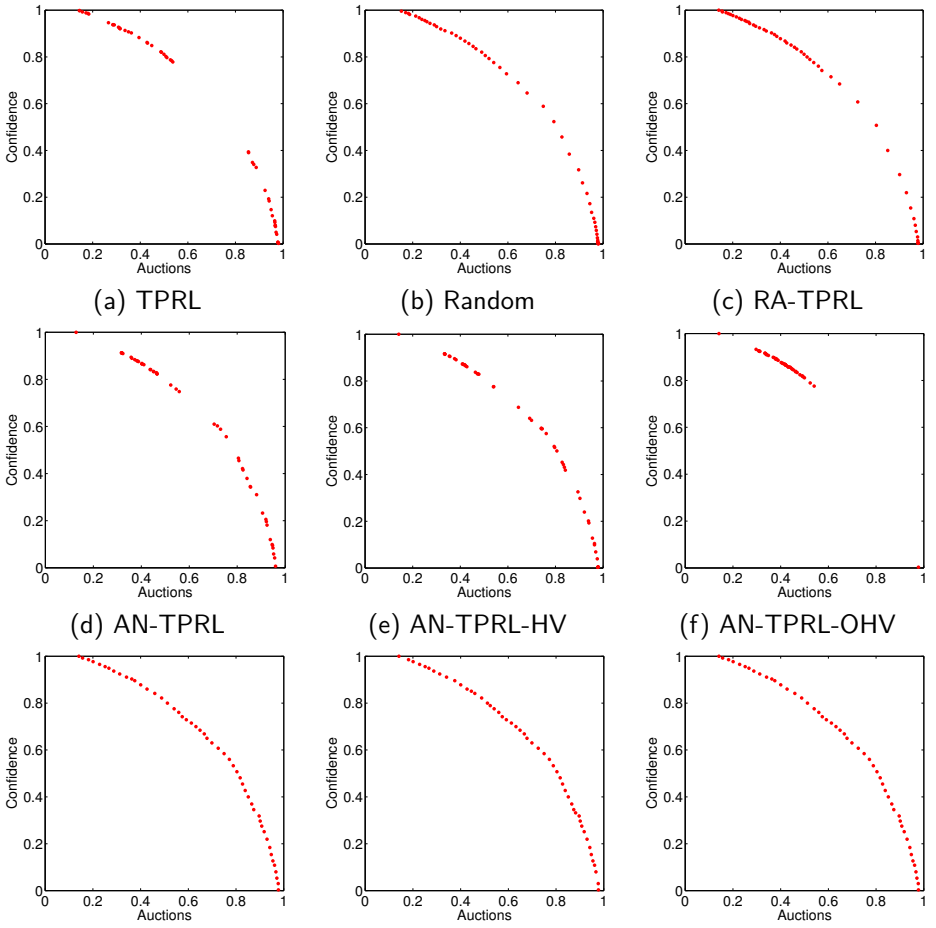


Figure 5.16: The hypervolume over time for each of the adaptive weight algorithms on scenario 1.

In Figure 5.16 we analyse the *anytime* property of each method, i.e., how fast does the algorithm explore the non-dominated parts of the Pareto front in terms of the hypervolume measure. We focus on scenario 1, but the conclusions generalise to the other scenarios as well. Note that each iteration represents the average value over 10 episodes with a specific scalarisation, determined by an AWA. One episode itself consists of 1000 simulations steps. The uniform distribution in the weight space, i.e., TPRL, is the original AWA used in Lewis et al. (2013) on this problem. We clearly see that the naive methods such as TPRL, random and RA-TPRL explore the objective space quite slowly in terms of the hypervolume measure. The adaptive methods such as AN-TPRL and AN-TPRL-HV, which adapt their weights by considering the 'gap' between solutions, perform roughly the same as their lines overlap. We note that their hypervolume values increase rapidly in early stages, while they stagnate after 25 iterations. In the end, the performance approaches RA-TPRL and randomly exploring the weight space.

In Figure 5.17 (a) to (e), we denote the final archive obtained by each of the methods. We normalised the values of the solutions for each of the methods and transformed them in order to create a maximisation problem for both objectives. We note that some methods are better at dividing the computational effort across the objective space. For example, Figure 5.17 (a) is a clear indication that a uniform distribution of the weight space with TPRL does not guarantee a uniform spread in the objective space. AN-TPRL and AN-TPRL-HV, in Figure 5.17 (d) and (e), adapt the weights in terms of the Euclidean and hypervolume metric, respectively. However, those algorithms focus their resources on particular areas, while leaving other, possibly interesting, trade-off solutions uncharted. In the following section, we highlight the reasons for the limited coverage of the objectives space of these methods.



(a) TPRL (b) Random (c) RA-TPRL

(d) AN-TPRL (e) AN-TPRL-HV (f) AN-TPRL-OHV

(g) RA-TPRL-DIST (h) RA-TPRL-HV (i) RA-TPRL-OHV

Figure 5.17: The Pareto fronts obtained after 50 iterations for each of the adaptive weight algorithms on Scenario 1.

5.3.4 Novel adaptive weight algorithms for RL

Although the adaptive weight algorithms have served their purpose on a number of test domains (Dubois-Lacoste et al., 2011; Lust and Teghem, 2010), there are still some open questions on their performance in MORL. This doubt is based on the following reasons.

Firstly, the AWAs of Section 5.3.2 are tailored for stochastic local search algorithms that can be seeded using particular solutions to provide the optimisation algorithm with a good initial position in the search space. The AWAs make extensive use of this property to bias the search direction to fruitful and uncharted areas of the Pareto front. However, seeding the reinforcement learning agent in such a way is not possible as most problems tend to be episodic and consist of multiple stages that the agent has to go through. Secondly, the dichotomic scheme in Equation 5.5 uses the segment between two solutions s_1 and s_2 to calculate the new weight. However, the equation does not guarantee that the resulting solution obtained by the calculated weight will lie between the interval of the two parent solutions, i.e., between s_1 and s_2.

The solution that we propose is to combine the properties of RA-TPLS and AN-TPLS, meaning that we use the layered-division of the weight space where deeper layers intensify the search process to particular areas of RA-TPLS together with an adaptive ordering of the elements of the different layers, based on the aspects of the Pareto front currently being explored. Similarly to AN-TPLS we use different metrics, such as the Euclidean distance and the hypervolume indicator, to qualify the difference between solutions. By merging both procedures, we combine the best of both worlds, i.e., the layered approach allows us to explore the rough outline of the Pareto front in initial iterations of the algorithm and secondly, we do not rely on the dichotomic scheme in Equation 5.5 which does not scale well to general application domains. Subsequently, we no longer require specific seeds that bias the search direction and eventually also the performance of the obtained solutions. We call this algorithm *RA-TPRL-DIST* and *RA-TPRL-HV* with the Euclidean and hypervolume metric, respectively. An outline of the procedure for the bi-objective case is given in Algorithm 4. The algorithms starts by examining the bounds of the Pareto front, i.e., we evaluate the optimisation algorithm, in this case $RL()$, with weights $w_1 = 0$ and $w_1 = 1$. The solutions s_1 and s_2 and their corresponding weights are added to a tree-like data structure that stores the segments of the weight space (line 4). Every iteration, the largest segment, according to a metric (Euclidean distance or hypervolume), that is not explored yet is determined (line 6) and the new weight is calculated to be in the middle of that segment (line 7). Subsequently, the reinforcement learning algorithm is run with that weight and 2 new segments are added to the tree. i.e., the segment connecting the left solution and the middle solution and another segment from the middle to the right solution. (lines 9 and 10). Eventually, the *Archive* stores the set of multi-objective solution acquired throughout the run.

We also see opportunities for trying out an alternative metric. The hypervolume measure in AN-TPLS-HV is an interesting indicator that calculates the surface that two solutions occupy in the two-dimensional objective space. Although the hypervolume measure in Equation 5.6 works well for calculating the volume of two solutions in the objective space,

Algorithm 4 An Improved AWA

1: $s_1 \leftarrow RL(0)$
2: $s_2 \leftarrow RL(1)$
3: Add s_1, s_2 to *Archive*
4: tree $\leftarrow$ new Tree(new Segment$(0, s_1, 1, s_2)$)
5: **while** stopping criteria not met **do**
6: Segment lg $\leftarrow tree.getLargestGap()$
7: w $\leftarrow \frac{lg.weight_{s_1} + lg.weight_{s_2}}{2}$
8: $s' \leftarrow RL(w)$
9: tree.add(new Segment(lg.weight$_{s_1}$, lg.$_{s_1}$, w, s'))
10: tree.add(new Segment(w, s', lg.weight$_{s_2}$, lg.$_{s_2}$))
11: Add s' to *Archive*
12: **end while**
13: **return** *Archive*

it is not the common hypervolume formula. An alternative hypervolume calculation takes into account a particular reference point ref to measure the volume of a given set of solutions, for any number of elements. It is clear that some solutions in the set will have less or more contribution to the final performance because the rectangles, defined by the reference point to each of the solutions, will have significant overlaps. Therefore, it might be interesting to consider the degree of overlap between solutions directly and to minimise this overlap. We call this metric the *overlapping* hypervolume (OHV) measure (Figure 5.18). The measure calculates the ratio of the overlap and the unique hypervolume of the two solutions (Equation 5.9 and 5.13).The idea is to order the solutions using this measure in order to intensify the search to segments with the smallest overlap first.

Note that our test environment only considers two objectives, but we believe this algorithm could be generalised to more objectives quite easily, since both distance measures are applicable in m-dimensional spaces.

$$overlap = |(ref_1 - min(\mathcal{F}_1(s_1), \mathcal{F}_1(s_2)) \cdot (ref_2 - min(\mathcal{F}_2(s_1), \mathcal{F}_2(s_2))| \tag{5.9}$$

$$surf_{s_1} = |(\mathcal{F}_1(s_1) - ref_1) \cdot (\mathcal{F}_2(s_1) - ref_2)| \tag{5.10}$$

$$surf_{s_2} = |(\mathcal{F}_1(s_2) - ref_1) \cdot (\mathcal{F}_2(s_2) - ref_2)| \tag{5.11}$$

$$total = surf_{s_1} + surf_{s_2} - overlap \tag{5.12}$$

$$OHV(s_1, s_2) = \frac{overlap}{total} \tag{5.13}$$

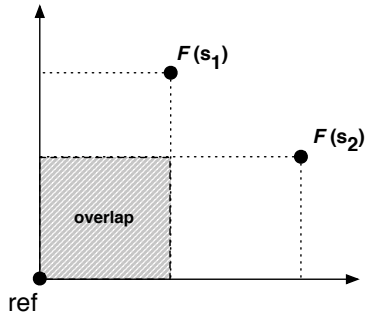


Figure 5.18: The overlapping hypervolume (OHV) measures the percentage of overlap between two solutions s_1 and s_2 in the objective space.

Results II

Next, we analyse and compare the former adaptive weight selection mechanisms of Section 5.3.2 and 5.3.4 on both simulated and real-life scenarios. We focus on the speed at which these methods approximate the Pareto front as measured by the quality indicators of Section 5.3.1.

First we investigate the anytime property of each of the methods. The results in Figures 5.19 extend the results of Figure 5.16 by including our novel methods. We note that the combination with the overlapping hypervolume metric (AN-TPRL-OHV) does not manage to improve a lot after the initial iterations as it reaches similar performance to TPRL on scenario 1. The best combination of speed of learning and final performance is obtained by our RA-TPRL-HV and RA-TPRL-OHV, while the Euclidean distance metric (RA-TPRL-DIST) learns significantly slower. For scenario 7 in Figure 5.20, similar conclusions can be formed except that AN-TPRL-OHV is now amongst the best performing algorithms, leaving TPRL far behind. In general, the naive methods and RA-TPRL-DIST are the slowest learners of the pack, while the adaptive methods reach similar performance in the end. In initial stages, the AN-TPRL and AN-TPRL-HV methods learn a bit faster, but this difference is negligible.

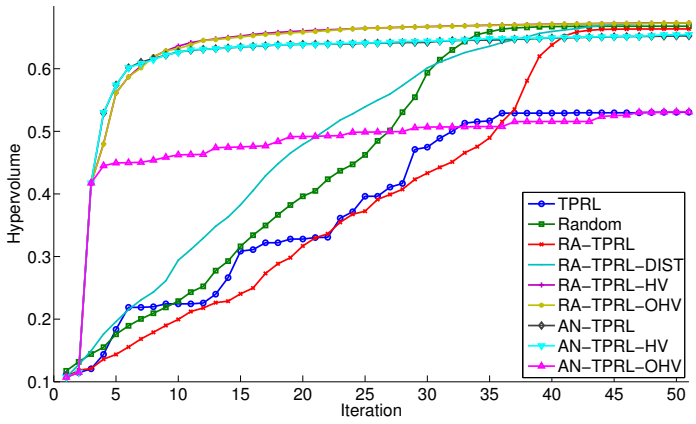


Figure 5.19: The hypervolume over time for each of the adaptive weight algorithms on scenario 1.

In Figure 5.17 the 50 solutions of the archive are depicted for each of the methods. We see that TPRL, although providing promising results in previous work on CamSim (Lewis et al., 2013), clearly lacks at exploring every part of the Pareto front as it leaves particular areas uncovered. On the third row of Figure 5.17, the methods that combine RA-TPRL with the distance, hypervolume and overlapping hypervolume metrics improve these results in terms of spread in the objective space, while minimising the gaps by taking into account the limited number of iterations. Other results using quality indicators that quantify these gaps in the Pareto front are presented in Table 5.1. We note that our RA-TPRL-HV method obtains the best results in terms of spread in the 10 scenarios. The best method in terms of generational (inverted) distance differ a lot for each scenario but in general the differences between the methods are minimal.

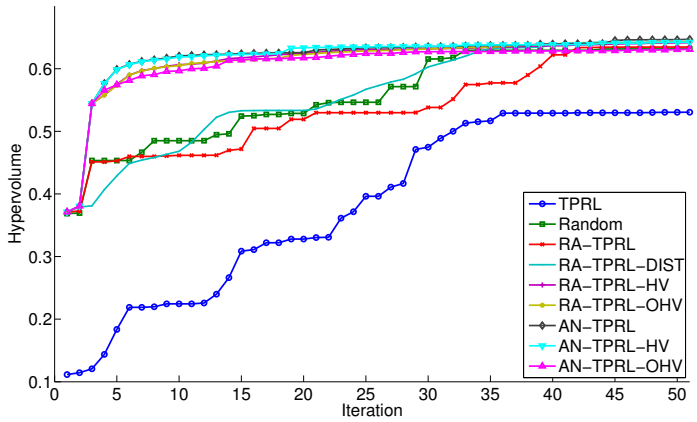


Figure 5.20: The hypervolume over time for each of the adaptive weight algorithms on scenario 7.

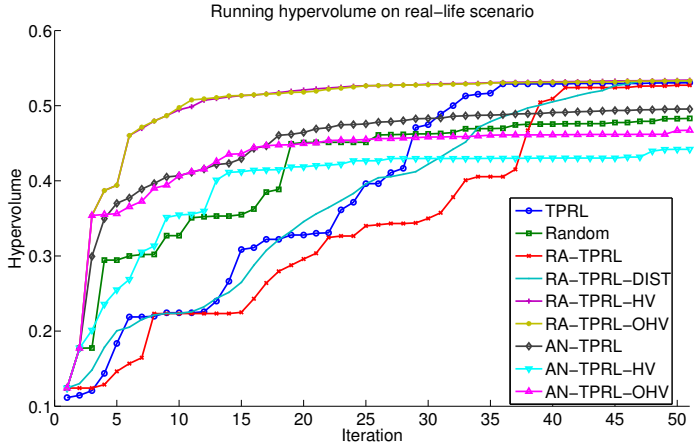


Figure 5.21: The hypervolume over time for each of the adaptive weight algorithms on the real-life data.

In Figure 5.21 the running hypervolume on the real-life data is depicted. Similar performance is obtained for the naive methods but in this example, we clearly see that the AN-TPRL methods are stuck at a specific performance level and cannot find contributions to their Pareto front. Our RA-TPRL-HV and RA-TPRL-OHV algorithms are able to improve these latter methods after 5 iterations. At the bottom of Table 5.1, we depict the results of the quality indicators of the different adaptive weight methods. We note that

the results are similar for the simulated data, meaning that best performance is obtained by the RA-TPRL-DIST, RA-TPRL-HV and RA-TPRL-OHV methods.

5.3.5 Conclusions

Environments with complex dynamics usually have a continuous, but finite-dimensional, Pareto front. For these situations, one can resort to heuristics to retrieve a good approximation set, containing solutions that are both qualitative and diverse. We have empirically analysed a series of algorithms that are based on a dichotomic scheme and we note that their performance could be improved significantly. Therefore, we propose a hierarchical decomposition of the weight space in combination with different metrics. These metrics determine the order in which the elements are to be evaluated. All the methods are extensively compared on 11 network scenarios in the multi-objective multi-agent CamSim framework. On these problems, we note that our methods can (1) explore the Pareto front faster, (2) obtain improved solutions in terms of hypervolume and (3) better spread in the objective space. Nevertheless, these algorithms remain heuristics and they do not provide guarantees for all possible multi-objective problems. Therefore, there is always room for improvement and need for other adaptive weight algorithms that use different metrics to explore the objective space.

5.4 Summary

In this chapter, we investigated mechanisms that allow single-policy algorithms to iteratively obtain multiple policies. By repeatedly running a single-policy algorithm with different weight configurations, one can attempt to find a set of high-quality policies. We have seen that for small and discrete Pareto fronts, one can employ tree-based search strategies that efficiently refine the search space in a divide-and-conquer style. For continuous Pareto fronts, the set of optimal policies is in essence infinite and one needs to come up with a good approximation set. This approximation set should contain policies of high-quality and diversity in the objective space. For this situation, we have assessed the performance of several established adaptive weight algorithms to highlight their shortcomings. In response to that, we have proposed a hierarchical decomposition of the weight space in combination with different metrics.

	TPRL	Random	RA-TPRL	RA-TPRL-DIST	RA-TPRL-HV	RA-TPRL-OHV	AN-TPRL	AN-TPRL-HV	AN-TPRL-OHV
Scenario 1									
$GSpread$	0.97501	0.91959	0.95257	0.80777	0.81487	**0.8076**	1.00112	1.00059	1.04571
HV	0.60384	0.66785	0.66347	0.6732	**0.67332**	0.67311	0.65238	0.65446	0.53106
$GDistances$	2.99143	2.98875	2.96057	2.92172	**2.91005**	2.92752	2.92267	2.95842	2.94144
$IGDistance$	14.20181	14.07198	14.07761	14.07234	**14.07109**	14.07234	14.07971	14.07675	14.14254
Scenario 2									
$GSpread$	0.90053	0.92837	0.92312	0.93271	**0.87589**	0.92952	0.95014	1.00981	0.91466
HV	0.66331	0.66783	0.67734	0.67008	0.67916	0.67087	0.6792	**0.68073**	0.65541
$GDistances$	2.83976	2.82884	2.74931	2.9209	2.87788	2.91765	3.02818	3.1312	**2.74639**
$IGDistance$	12.95593	12.95081	**12.90369**	12.95266	12.97541	12.95451	12.96075	12.98492	12.93109
Scenario 3									
$GSpread$	0.89829	0.92254	0.88001	0.90019	**0.85584**	0.88654	0.95599	1.05615	0.97401
HV	0.55002	0.54008	0.54899	0.54764	0.5553	0.54865	**0.58074**	0.53527	0.52907
$GDistances$	3.03947	3.04809	**2.98084**	2.99027	2.9913	2.99857	3.10163	3.20551	3.02713
$IGDistance$	**13.99183**	14.1615	14.03292	14.2017	14.10126	14.18741	14.08196	14.17774	14.05668
Scenario 4									
$GSpread$	0.90403	0.92403	0.92751	0.88937	**0.87291**	0.87961	0.9389	1.02157	0.96905
HV	0.46702	0.45717	0.46915	0.47347	0.47403	**0.47481**	0.46587	0.45524	0.44411
$GDistances$	3.11392	3.11126	**3.06278**	3.08443	3.08056	3.07423	3.15675	3.22976	3.15611
$IGDistance$	14.81707	**14.7147**	14.8793	14.84748	14.89994	14.81748	14.85078	14.85685	14.88275
Scenario 5									
$GSpread$	0.89728	0.93505	0.93387	0.8638	**0.83163**	0.85699	0.9592	1.02245	0.98591
HV	0.7239	0.70845	0.73191	0.73584	**0.73768**	0.73611	0.72231	0.71118	0.61485
$GDistances$	2.8878	2.8815	2.84129	**2.8124**	2.81807	2.83094	3.00157	3.06383	2.94756
$IGDistance$	13.20978	13.28262	**13.16021**	13.16986	**13.16021**	13.17839	13.21716	13.26455	13.31449
Scenario 6									
$GSpread$	0.89975	0.90772	0.90297	0.92098	**0.8757**	0.92804	0.95141	1.04161	0.91413
HV	**0.35623**	0.31604	0.33999	0.32833	0.34209	0.32689	0.34858	0.33307	0.32476
$GDistances$	3.23164	3.4249	**3.15232**	3.27144	3.36656	3.40051	3.49461	3.61006	3.37513
$IGDistance$	14.84423	**14.60966**	14.63511	14.63258	14.63511	14.62701	14.63511	14.63511	14.62822
Scenario 7									
$GSpread$	0.88941	0.91815	0.91261	0.87262	**0.86677**	0.87676	0.92367	1.02515	0.92114
HV	0.6351	0.63149	0.63461	0.64238	0.64151	0.64271	**0.64698**	0.64234	0.63078
$GDistances$	2.91725	2.91555	**2.85605**	2.89079	2.88499	2.8822	2.98927	3.08699	2.94745
$IGDistance$	13.56464	13.60138	13.62932	13.60277	13.60277	13.62932	13.52138	13.56219	**13.50574**
Scenario 8									
$GSpread$	0.89045	0.89952	0.92083	0.87468	**0.8426**	0.85821	0.94294	1.00346	0.90937
HV	0.67929	0.67015	**0.70277**	0.69972	0.69903	0.70077	0.68043	0.6726	0.64769
$GDistances$	2.87852	2.86679	**2.81096**	2.86095	2.87561	2.858	2.99304	3.0593	2.90116
$IGDistance$	13.32639	**13.12466**	13.25595	13.29731	13.2644	13.27501	13.23966	13.25307	13.27036
Scenario 9									
$GSpread$	0.98498	0.97623	0.9948	0.85214	**0.84111**	0.8635	1.01375	1.00589	1.01342
HV	0.77092	0.72027	0.76646	0.77816	**0.77854**	0.77773	0.75198	0.74881	0.71142
$GDistances$	2.86232	2.87269	2.8102	2.76207	**2.755**	2.77794	2.84333	2.90014	2.92109
$IGDistance$	13.12838	13.17884	13.11477	13.11224	**13.11206**	13.11371	13.15331	13.17327	13.18997
Scenario 10									
$GSpread$	0.94538	0.97646	0.95739	0.94302	**0.91827**	0.93815	1.00365	1.06029	1.00073
HV	0.62974	0.63441	0.63944	0.63959	0.63967	0.63957	0.63652	**0.64938**	0.63766
$GDistances$	2.80424	2.79834	**2.7291**	2.73846	2.75131	2.73696	2.95668	3.00213	2.82691
$IGDistance$	12.79585	**12.72557**	12.74433	12.74441	12.74441	12.74441	12.74441	12.74229	12.73365
Scenario 11									
$GSpread$	0.94689	0.96702	0.95472	0.92537	**0.90926**	0.91007	0.98978	1.05081	0.99617
HV	0.56738	0.56372	0.56865	0.57189	0.57219	0.57197	0.5778	**0.58042**	0.56247
$GDistances$	2.94136	2.94263	**2.88296**	2.90703	2.91029	2.90481	2.99781	3.05835	3.00429
$IGDistance$	13.93518	**13.93155**	13.93476	13.93734	13.93801	13.93734	13.99275	13.95814	13.95763
Real-life scenario									
$GSpread$	0.8974	0.90369	0.89542	0.86498	0.85507	**0.85358**	0.94972	1.016	0.95567
HV	0.53053	0.48302	0.52724	0.53287	**0.53401**	0.5332	0.49554	0.44196	0.46728
$GDistances$	3.10149	3.12013	3.09672	**3.05587**	3.05963	3.06153	3.09171	3.17102	3.11861
$IGDistance$	14.9041	15.24093	14.98453	**14.77806**	**14.77806**	**14.77806**	14.96315	15.17256	15.12926

Table 5.1: Quality indicators on 12 different scenarios in CamSim

6 | Simultaneously learning multiple multi-objective policies

In the previous two chapters, we focussed our attention on single-policy reinforcement learning algorithms, i.e., algorithms that obtain a single policy at each run. Performing repeated runs of these algorithms might act as a suitable but primitive approach to discover a set of trade-off solutions. In this chapter, we investigate algorithms that retrieve various compromise solutions simultaneously. Hence, they are called *multi-policy* algorithms. In this chapter, we propose two multi-policy algorithms that each cover a particular topic of MORL.

A first algorithm is the *multi-objective hierarchical heuristic optimisation* (MO-HOO) for multi-objective $\mathcal{X}$-armed bandit problems. This is a class of problems similar to a standard bandit problem. Only in this case, the agent is faced with a continuous, finite-dimensional action space instead of a discrete one. MO-HOO evolves a tree of Pareto non-dominated estimates that explores the continuous action space in a hierarchical manner. As a result, the algorithm is specifically tailored for multi-objective industrial optimisation problems that involve the assignment of continuous variables.

A second algorithm is *Pareto Q-learning*. Pareto Q-learning (PQL) combines the principles of MO-HOO with Q-learning to make it applicable in multi-state problems. PQL is the first temporal difference-based multi-policy MORL algorithm that does not use the linear scalarisation function. Pareto Q-learning is not limited to the convex hull, but it can learn the entire Pareto front of deterministic non-stationary policies, if enough exploration is provided. The algorithm learns the Pareto front by bootstrapping Pareto non-dominated $\mathbf{Q}$-vectors throughout the state and action space. As we will see, one of the crucial aspects of this algorithm is its bootstrapping process, i.e., how to update a set of $\mathbf{Q}$-vectors

with another set of $\mathbf{Q}$-vectors, bearing in mind that both sets might not contain the same number of elements.

In more detail, we will present the following contributions in this chapter:

- Analyse related work on multi-policy MORL
- Propose and evaluate a multi-policy MORL algorithm for multi-objective $\mathcal{X}$-armed bandit problems
- Propose and evaluate a multi-policy temporal-difference MORL algorithm for multi-objective sequential decision making problems

This research has been published in Van Moffaert et al. (2014b), Van Moffaert and Nowé (2014) and Van Moffaert et al. (2015).

6.1 Related work on simultaneously learning multiple policies

Over the years, several algorithms have been proposed that simultaneously retrieve multiple multi-objective policies. Usually, these algorithms fall into two separate categories, i.e., algorithms that employ either linear scalarisation functions or monotonically increasing scalarisation functions, such as the Pareto dominance relation.

Most algorithms that learn multiple policies belong to the first category. Because of the properties of linear scalarisation functions, sum and union operators can be quite straightforwardly defined on convex hulls. One of the most important contributions is the *convex hull value-iteration* (CHVI) algorithm which computes the deterministic stationary policies that lie on the convex hull of the Pareto front (Barrett and Narayanan, 2008). From batch data, CHVI extracts and computes every linear combination of the objectives in order to obtain all deterministic optimal policies. CHVI bootstraps by calculating the convex hull of the union over all actions in s', that is $\bigcup_{a'} \mathbf{Q}(s', a')$. The most computationally expensive operator is the procedure of combining convex hulls in the bootstrapping rule. Lizotte et al. (2010) reduce the asymptotic space and time complexity of the bootstrapping rule by simultaneously learning several value functions corresponding to different weights and by calculating their piecewise linear spline representation. They validated their work on clinical trial data for two objectives to propose a treatment to patients based on a trade-off between the effectiveness of the drugs and severity of the side effects. Nevertheless, the practical possibilities for higher dimensional spaces are not straightforward (Lizotte et al., 2012). In Kazuyuki et al. (2009), the *parallel Q-learning* algorithm is proposed. Parallel Q-learning is similar to CHVI in the sense that it learns optimal piecewise-linear policies for all weights. It computes the convex hull by defining sum and union operators for sets, similar to the way CHVI bootstraps. The algorithm suffers from convergence issues since vertices are continuously being added to the polygon representing the convex hull. To overcome this, the authors introduce a threshold parameter to reduce the size of the sets and to increase the accuracy. In Castelletti et al. (2011, 2012), the fitted Q-iteration

algorithm is extended to learn sets of policies in multi-objective environments. This off-line algorithm is called multi-objective fitted Q-iteration (MOFQI) and it computes the set of expected return vectors obtained for a series of weight vectors in a single run.

The second category comprises algorithms that employ monotonically increasing functions which are more general than the linear scalarisation function. A typical member of this class would be the Pareto dominance relation. How it can be combined into a learning paradigm has been a subject of research for many decades. In White (1982), a dynamic programming algorithm simultaneously computes a set of Pareto non-dominated policies, i.e., a $\hat{Q}_{set}$. The algorithm bootstraps the Pareto non-dominated $\mathbf{Q}$-vectors of the next state to the set of the current state-action pair. The idea is that, after the discounted Pareto non-dominated rewards are propagated and the $\hat{Q}_{set}$'s converge to a set of Pareto non-dominated policies, the user can traverse the tree of $\hat{Q}_{set}$'s by applying a preference function. Wang and Sebag (2012, 2013) propose a multi-objective Monte Carlo tree search (MO-MCTS) algorithm to learn a set of solutions. In MCTS, a search tree is incrementally built and explored at the same time (Coulom, 2007). The nodes of the tree represent visited states and branches represent actions. At the end of an episode, the nodes are weighted according to the outcome of the episode in order to bias the action selection in future plays. In MO-MCTS, the upper confidence bounds of the actions are scalarised in either of two distinct manners. One possibility is to apply the hypervolume quality indicator to determine which path of vectorial estimates in the tree introduces the largest hypervolume contribution w.r.t. the policies that are already retrieved. Because the hypervolume measure is costly to compute, as an alternative, it is also possible to simply determine whether or not a tree walk obtained a non-dominated return in a boolean fashion. This way, a scalarised multi-objective value function is constructed that eases the process of selecting an action with vectorial estimates.

As highlighted by Roijers et al. (2013), there is a specific need for on-line reinforcement learning algorithms that are able to discover the Pareto front although the setting is far from straightforward. To fill this gap, we will research this problem from two distinct viewpoints. Moreover, in the case of (1) single-state problem involving a continuous but finite dimensional action space in Section 6.2 and (2) for multi-state sequential decision making problems in Section 6.3.

6.2 Learning multiple policies in single-state problems

In the classical bandit problem, described in Section 2.5, a gambler wants to maximise his reward by selecting the best possible action from a finite set of arms with unknown reward distributions. The on-line aspect of this problem is very important, i.e. in order to maximise his gain, the gambler needs to find a balance between exploring uncharted territory and exploiting his current knowledge.

Recently, this problem is generalised to environments where the action space is continuous but finite-dimensional. In this case, the gambler is faced with an continuous set of arms. This problem is called an $\mathcal{X}$-armed bandit problem $\mathcal{B}$ consisting of a pair of

$\mathcal{B} = (\mathcal{X}, M)$, where $\mathcal{X}$ is a measurable space of arms and M determines the distribution of rewards associated with each arm (Bubeck et al., 2010). This problem has a strong resemblance to operational research and control applications that involve the calibration of several parameters. Examples of such applications are the tuning of hydrostatic drive-train controllers (Van Vaerenbergh et al., 2014) and the adjustment the temperature levels to maximise a chemical reaction (Cope, 2009). One of the algorithms that effectively solves a $\mathcal{X}$-armed bandit problem is the *hierarchical optimistic optimisation* (HOO) strategy (Bubeck et al., 2010) which will be explained in the subsequent section.

6.2.1 Hierarchical Optimistic Optimisation

The HOO strategy iteratively constructs a binary tree over the action space $\mathcal{X}$. A node in the binary tree represents an area over $\mathcal{X}$ and is represented by (h, i), where i is the index of the node at depth h. Hence, the root node is represented by $(0, 1)$. Following the terminology of Bubeck et al. (2010), $(h+1, 2i-1)$ and $(h+1, 2i)$ refer to the child nodes of the node located at (h, i). Let $\mathcal{P}_{h,i} \subset \mathcal{X}$ be the area of the action space corresponding to node (h, i), given these two conditions:

$$\mathcal{P}_{0,1} = \mathcal{X} \tag{6.1}$$

$$\mathcal{P}_{h,i} = \mathcal{P}_{h+1,2i-1} \cup \mathcal{P}_{h+1,2i}, \forall h \geq 0 \text{ and } 1 \leq i \leq 2^h. \tag{6.2}$$

A node stores an optimistic estimate of the quality of its subtree, i.e. a $\mathcal{U}$-value. The $\mathcal{U}_{h,i}(n)$-value is an initial estimate of the maximum value of the pay-off function in the corresponding region of the action space $\mathcal{X}$ and is defined as follows:

$$\mathcal{U}_{h,i}(n) = \begin{cases} \hat{\mu}_{h,i}(n) + \sqrt{\frac{2 \ln n}{\mathcal{T}_{h,i}(n)}} + \nu_1 \rho^h, \text{if } \mathcal{T}_{h,i}(n) > 0 \\ +\infty, \text{otherwise,} \end{cases}$$

where $\hat{\mu}_{h,i}(n)$ is the mean estimate of the subtree of node n and the other two terms express a standard optimistic exploration value and the maximum possible variation of the pay-off function, defined by the ν_1 and ρ^h parameters. When the action is not sampled before, it receives the highest optimistic value, i.e. $+\infty$. As we are interested in sampling near the maxima of the pay-off function, the HOO strategy focusses on estimating this function near its maxima while leaving the other, less interested parts of the function, less explored.

Given the $\mathcal{U}$-value of a node n, its $\mathcal{B}$-value is computed by taking into account its own $\mathcal{U}$-value and the $\mathcal{B}$-values of its two child nodes. The $\mathcal{B}$-value is designed to put a tighter upper-bound on the best possible value that can be obtained in that region of $\mathcal{X}$. The $\mathcal{B}$-value estimate is:

$$\mathcal{B}_{h,i}(n) \leftarrow \min \left\{ \mathcal{U}_{h,i}(n), \max\{\mathcal{B}_{h+1,2i-1}(n), \mathcal{B}_{h+1,2i}(n)\} \right\}. \tag{6.3}$$

Thus, the root node stores an (optimistic) estimate of the quality of the entire action space, where its left child stores a more accurate estimate for the interval $[\inf(\mathcal{X}), \frac{\sup(\mathcal{X})}{2}]$

of the action space, i.e. between the infimum of $\mathcal{X}$ and half of its supremum given the divide-and-conquer approach. Similarly, the right child of the root stores equivalent information for $[\frac{\sup(\mathcal{X})}{2}, \sup(\mathcal{X})]$. Selecting an action is performed by traversing the subtree with the largest $\mathcal{B}$-value until a leaf value is reached. Thereafter the action is sampled and the estimates of the nodes on the traversed path are updated and refined by considering the observed reward and the number of times the subtree is visited in previous plays. An outline of the algorithm can be found in Algorithm 5.

Bubeck et al. (2010) prove this algorithm to converge to the mean pay-off function around its maxima if the function's smoothness is conform to being locally *Lipschitz* continuous. In short, a function is said to be Lipschitz continuous if a small change in the function's input values signifies a mild change in the corresponding function values of that function, which is a reasonable assumption in many engineering problems.

6.2.2 Multi-Objective Hierarchical Optimistic Optimisation

Transforming the single-objective HOO strategy to multi-objective environments requires several adjustments to the algorithm's internal workings. Below, we tackle the required modifications for constructing the Multi-Objective Hierarchical Optimistic Optimisation (MO-HOO) algorithm into four different categories.

Sampling purpose

A first crucial aspect that arises when transforming a single-objective on-line algorithm to a multi-objective environment is defining its purpose. In the standard problem the goal of the agent is to maximise the average reward being received, which is the logical approach as there is a total order between the scalar rewards. In the case of multiple objectives, the Pareto dominance relation only supplies a partial order, i.e. only non-dominated and dominated vectors can be compared but non-dominated vectors are incomparable amongst each other. As a consequence, if one would average the reward vectors being obtained by the agent, we are not guaranteed that the average is an actual attainable solution for the problem. Take for instance, a very easy multi-objective problem where the agent can only take two action a and b where the deterministic rewards are $(1, 0)$ and $(0, 1)$, respectively. If one would take action a with probability x and action b with probability $p = (1 - x)$, the average reward vector would be $(x, 1 - x)$. Thus, although there are only two possible outcomes for the problem, considering average reward vectors implicates that we are no longer sampling on the Pareto front but on the convex hull of the Pareto front, similar to the stochastic mixture policies of Section 3.4.1. Therefore, averaging vectorial rewards does not represent actual solution points in the objective space.

In control problems we are not interested in stochastic mixture policies obtained from the convex hull of the Pareto front, but rather in sampling a genuine trade-off in every execution of the policy. This means that we want to identify actually realisable control policies that lie on the Pareto front that offer a fixed trade-off between the different control objectives. However, despite the fact that our main goal is to identify the Pareto front, we

Algorithm 5 The HOO strategy

Parameters: Two real numbers $\nu_1 > 0$ and $\rho \in (0,1)$, a sequence $(\mathcal{P}_{h,i})_{h\geq 0, 1\leq i\leq 2^h}$ of subsets of $\mathcal{X}$ satisfying the conditions (6.1) and (6.2).

Auxiliary function $\textsc{Leaf}(\mathcal{T})$: outputs a leaf of $\mathcal{T}$.

Initialization: $\mathcal{T} = \big\{(0,1)\big\}$ and $B_{1,2} = B_{2,2} = +\infty$.

for $n = 1, 2, \ldots$ **do** ▷ Strategy HOO in round $n \geq 1$
  $(h,i) \leftarrow (0,1)$ ▷ Start at the root
  $P \leftarrow \{(h,i)\}$ ▷ P stores the path traversed in the tree
  while $(h,i) \in \mathcal{T}$ **do** ▷ Search the tree $\mathcal{T}$
    if $B_{h+1,2i-1} > B_{h+1,2i}$ **then** ▷ Select the "more promising" child
      $(h,i) \leftarrow (h+1, 2i-1)$
    else if $B_{h+1,2i-1} < B_{h+1,2i}$ **then**
      $(h,i) \leftarrow (h+1, 2i)$
    else ▷ Tie-breaking rule
      $Z \sim \mathrm{Ber}(0.5)$ ▷ e.g., choose a child at random
      $(h,i) \leftarrow (h+1, 2i-1)$
    end if
    $P \leftarrow P \cup \{(h,i)\}$
  end while
  $(H,I) \leftarrow (h,i)$ ▷ The selected node
  Choose arm X in $\mathcal{P}_{H,I}$ and play it ▷ Arbitrary selection of an arm
  Receive corresponding reward Y
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{(H,I)\}$ ▷ Extend the tree
  for all $(h,i) \in P$ **do** ▷ Update the statistics T and $\hat{\mu}$ stored in the path
    $T_{h,i} \leftarrow T_{h,i} + 1$ ▷ Increment the counter of node (h,i)
    $\hat{\mu}_{h,i} \leftarrow \big(1 - 1/T_{h,i}\big)\hat{\mu}_{h,i} + Y/T_{h,i}$ ▷ Update the mean $\hat{\mu}_{h,i}$ of node (h,i)
  end for
  for all $(h,i) \in \mathcal{T}$ **do** ▷ Update the statistics U stored in the tree
    $U_{h,i} \leftarrow \hat{\mu}_{h,i} + \sqrt{(2\ln n)/T_{h,i}} + \nu_1 \rho^h$ ▷ Update the U–value of node (h,i)
  end for
  $B_{H+1,2I-1} \leftarrow +\infty$ ▷ $\mathcal{B}$–values of the children of the new leaf
  $B_{H+1,2I} \leftarrow +\infty$
  $\mathcal{T}' \leftarrow \mathcal{T}$ ▷ Local copy of the current tree $\mathcal{T}$
  while $\mathcal{T}' \neq \big\{(0,1)\big\}$ **do** ▷ Backward computation of the $\mathcal{B}$–values
    $(h,i) \leftarrow \textsc{Leaf}(\mathcal{T}')$ ▷ Take any remaining leaf
    $B_{h,i} \leftarrow \min\Big\{U_{h,i},\, \max\big\{B_{h+1,2i-1}, B_{h+1,2i}\big\}\Big\}$ ▷ Backward computation
    $\mathcal{T}' \leftarrow \mathcal{T}' \setminus \big\{(h,i)\big\}$ ▷ Drop updated leaf (h,i)
  end while
end for

do not want to neglect the sampling efficiency of our method. Our motivating assumption here is that sampling policies inherently has a certain cost associated with it, either in time, in resources or both. Often these costs are associated with evaluating a control strategy on a real system. Furthermore, we assume that solutions which lie far from the Pareto front are suboptimal in at least one of the objectives and thus typically have higher costs associated with them, e.g., because they are less efficient, they take longer to reach a goal, or they might even violate safety constraints. Therefore, our goal is to develop an efficient method for identifying the Pareto front, which minimises the amount of sampling performed in regions far from the Pareto front.

Averaging over subtrees

In the original implementation of the HOO strategy, the $\mathcal{U}$-value of a node stores an estimate of the average reward of a subtree together with some additional statistics. However, as we are dealing with multiple objectives at the same time, there is no point in averaging multiple reward vectors into a single value for each objective. Together with the fact that in the HOO strategy, one does not sample the same action more than once, but rather its left or right child node, we no longer consider the $\mathcal{U}$-values in their original form. Therefore, in the multi-objective case, we only propagate the $\boldsymbol{\mathcal{U}}$-vector of leaf nodes as they are samples that are not averaged out over subtrees.

Knowledge representation

Recall that in the single-objective HOO strategy, the agent's goal is to maximise the scalar reward obtained by sampling close to the optimum of the pay-off function. Therefore, the algorithm propagates an estimate of the maximum of the pay-off function from leaf nodes to the root node in terms of $\mathcal{B}$-values. In a multi-objective problem, however, there usually is no single-optimum but there are multiple Pareto non-dominated solutions. Thus, a scalar estimate or $\mathcal{B}$-value is insufficient to store the quality of a region of the pay-off function. The solution we propose is to consider sets of estimates, i.e., $\mathcal{B}$-sets. The elements of the sets are $\boldsymbol{\mathcal{B}}$-vectors that store an estimate for each objective. The most optimistic upper bound is now a vector $\mathbf{u}$ that is always a member of the Pareto front without excluding other Pareto optimal vectors, e.g. $\mathbf{u} = (-\infty, +\infty)$ for an environment with two objectives.

Information propagation

In Equation 6.3, the HOO strategy performs a backward computation to adjust the estimate of a node by taking into account its $\mathcal{U}$-value and the $\mathcal{B}$-value of its two child nodes. To determine the $\mathcal{B}$-value of a node, the algorithm determines the highest upper bound of its child nodes by a $\max$ operator. In the multi-objective variant, we replace this operator by the operator $ND(\bigcup_{\boldsymbol{\mathcal{B}}}(\boldsymbol{\mathcal{B}}_{h+1,2i-1}(n), \boldsymbol{\mathcal{B}}_{h+1,2i}(n))$ which yields every non-dominated $\mathcal{B}$-vector of the left and right child of node n. The $\min$ operator in Equation 6.3 assures a tighter bound of the $\mathcal{B}$-value of node n. As we are not focussing on the average Pareto

front or convex hull, we no longer consider the $\mathcal{U}$-value of non-leaf nodes. However, as the $\boldsymbol{\mathcal{U}}$-vectors of leafs are recalculated at every iteration, the additional statistics introduce a level of robustness in the case of noisy environments.

If one would not be interested in the actual trade-off achieved in every execution of the policy, but rather in the average sampling performance, a multi-objective $\min$ operator should be proposed. The specification of a multi-objective $\min$ operator could place a stricter bound on the $\boldsymbol{\mathcal{B}}$-vectors in order to make them less optimistic. However, defining such a $\min$ operator that would work in a multi-objective setting with multiple incomparable solutions is currently an open question as it is not clear to determine the logical outcome of applying a $\min$ operator on a set of non-dominated $\boldsymbol{\mathcal{B}}$-vectors of child nodes and a $\boldsymbol{\mathcal{U}}$-vector since there is no total order.

An algorithmic outline of the MO-HOO strategy for an m-objective problem can be found in Algorithm 6.

Algorithm 6 The MO-HOO strategy

Parameters: Two real numbers $\nu_1 > 0$ and $\rho \in (0,1)$, a sequence $(\mathcal{P}_{h,i})_{h\geq 0, 1\leq i\leq 2^h}$ of subsets of $\mathcal{X}$ satisfying the conditions (6.1) and (6.2).

Auxiliary function $\textsc{Leaf}(\mathcal{T})$: outputs a leaf of $\mathcal{T}$.

Initialisation: $\mathcal{T} = \big\{(0,1)\big\}$ and $\boldsymbol{\mathcal{B}}_{1,2} = (+\infty, \ldots, -\infty)$ and $\boldsymbol{\mathcal{B}}_{2,2} = (-\infty, \ldots, +\infty)$.

for $n = 1, 2, \ldots$ **do** ▷ Strategy MO-HOO in round $n \geq 1$
  $(h,i) \leftarrow (0,1)$ ▷ Start at the root
  $P \leftarrow \{(h,i)\}$ ▷ P stores the path traversed in the tree
  $\boldsymbol{\mathcal{B}}_{\text{rand}} \in \boldsymbol{\mathcal{B}}_{h,i}$ ▷ Select random Pareto dominant $\boldsymbol{B}$-vector
  while $(h,i) \in \mathcal{T}$ **do** ▷ Search the tree $\mathcal{T}$
    if $\boldsymbol{\mathcal{B}}_{\text{rand}} \in \boldsymbol{B}_{h+1,2i-1}$ **then**
      $(h,i) \leftarrow (h+1, 2i-1)$
    else
      $(h,i) \leftarrow (h+1, 2i)$
    end if
    $P \leftarrow P \cup \{(h,i)\}$
  end while
  $(H,I) \leftarrow (h,i)$ ▷ The selected node
  Choose arm X in $\mathcal{P}_{H,I}$ and play it ▷ Arbitrary selection of an arm
  Receive corresponding reward vector $\mathbf{Y}$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{(H,I)\}$ ▷ Extend the tree
  for all $(h,i) \in P$ **do** ▷ Update the statistics T and $\hat{\mu}$ stored in the path
    $T_{h,i} \leftarrow T_{h,i} + 1$ ▷ Increment the counter of node (h,i)
    $\hat{\boldsymbol{\mu}}_{h,i} \leftarrow \big(1 - 1/T_{h,i}\big)\hat{\boldsymbol{\mu}}_{h,i} + \mathbf{Y}/T_{h,i}$ ▷ Update the mean vector $\hat{\boldsymbol{\mu}}_{H,I}$ of node (h,i)
  end for
  for all $(h,i) \in \mathcal{T}$ **do** ▷ Update the statistics U stored in the tree
    $\boldsymbol{U}_{h,i} \leftarrow \hat{\boldsymbol{\mu}}_{h,i} + \sqrt{(2\ln n)/T_{h,i}} + \nu_1 \rho^h$ ▷ Update the U–value set of node (h,i)
  end for
  $\boldsymbol{B}_{H+1,2I-1} \leftarrow (+\infty, \ldots, -\infty)$ ▷ m-ary incomparable $\boldsymbol{B}$–vectors to leaf's children
  $\boldsymbol{B}_{H+1,2I} \leftarrow (-\infty, \ldots, +\infty)$
  $\mathcal{T}' \leftarrow \mathcal{T}$ ▷ Local copy of the current tree $\mathcal{T}$
  while $\mathcal{T}' \neq \big\{(0,1)\big\}$ **do** ▷ Backward computation of the $\mathcal{B}$–values
    $(h,i) \leftarrow \textsc{Leaf}(\mathcal{T}')$ ▷ Take any remaining leaf
    if $\boldsymbol{B}_{h,i}$ *is* $\textsc{Leaf}(\mathcal{T})$ **then** ▷ If node is leaf in original tree
      $\boldsymbol{B}_{h,i} \leftarrow \boldsymbol{U}_{h,i}$
    else
      $\boldsymbol{B}_{h,i} \leftarrow ND \bigcup_{\boldsymbol{B}} \big(\boldsymbol{B}_{h+1,2i-1}, \boldsymbol{B}_{h+1,2i}\big)$ ▷ Backward computation
    end if
    $\mathcal{T}' \leftarrow \mathcal{T}' \setminus \big\{(h,i)\big\}$ ▷ Drop updated leaf (h,i)
  end while
end for

Computational complexity

The complexity is a crucial aspect of an algorithm as it determines its resources in terms of time and computational effort in relation to the size of the problem. In MO-HOO, the storage requirements are related to the size of the tree-structure. As in HOO (Bubeck et al., 2010), the size of the tree is at most n after n rounds. Therefore, the storage requirements are just $O(n)$.

In terms of computational effort, the algorithm loops over the tree at every round to update its statistics and to propagate the necessary information throughout the levels of the tree. This can be accomplished in $O(n)$ every round. As a result, the computational complexity of the algorithm is $O(n^2)$, where n is the number of rounds the algorithm is run.

6.2.3 Experiments

In the experimental section, we evaluate the performance of the MO-HOO algorithm on two typical multi-objective benchmark functions. These are the Schaffer 1 function and the Fonseca and Flemming function.

6.2.4 Schaffer 1 function

The first test function is the bi-objective Schaffer 1 function (Schaffer, 1985) that is defined by:

$$\text{maximise } f(x) = \begin{cases} f_1(x) = -x^2 \\ f_2(x) = -(x-2)^2, \end{cases}$$

where $x \in [-10, 10]$. Originally, these test functions are minimisation problems but we negate them to obtain a maximisation problem as this is what our reinforcement learning algorithm assumes. The function is depicted in Figure 6.1 where the Pareto front is convex and indicated by green dots.

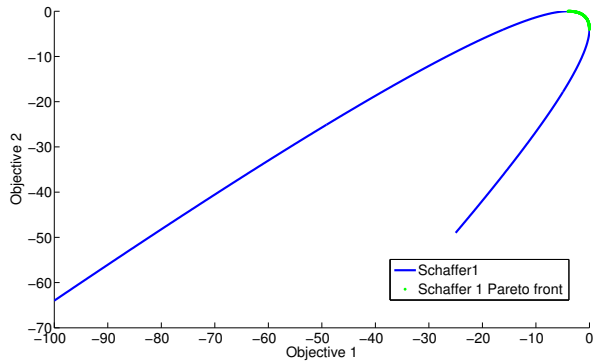


Figure 6.1: The objective space of the Schaffer 1 function. The Pareto front of the function is denoted by green dots.

In Figure 6.2 the sampling performance of the MO-HOO strategy is compared to a random sampling strategy. Recall that our goal is to sample as fast as possible as close as possible on the Pareto front. As the Pareto front is in this case continuous, we determine whether the vector valued return of the sampled action is within a distance of δ of a discrete approximation of the true Pareto front. In all our experiments, δ is set to 0.2 and the results are averaged over 50 independent runs. This discrete approximation of the true Pareto front is obtained by collecting the Pareto non-dominated outcomes of many independent random runs of the algorithm. From the figure, we note that the MO-HOO strategy gradually increases its performance by sampling closer to the Pareto front as the learning time increases. After 500 iterations, the probability of sampling a Pareto optimal action approaches 1. The random strategy only achieves a performance of 18% to 20%.

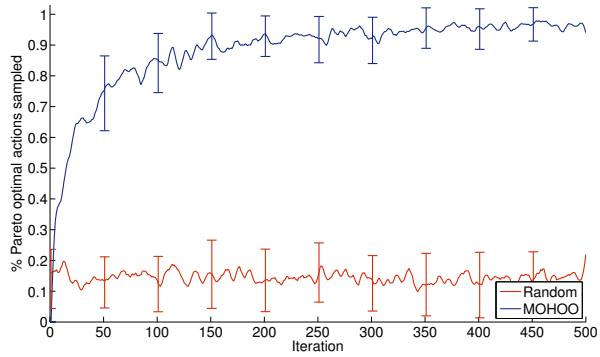


Figure 6.2: The sampling performance MO-HOO is compared to a naive random strategy. We note that the probability of the MO-HOO strategy of sampling Pareto optimal solutions approaches 1 as time increases.

In Figure 6.3, we plot three graphs that provide additional insights in the behaviour of the MO-HOO strategy. In a first subplot, we denote the sampled action at each time step in the normalised action space. The MO-HOO strategy is exploring in early stages of the process and afterwards it is exploiting its knowledge quite rapidly. In a second subplot, we show the depth of the tree in relation to the action space. Subtrees that are more expanded have a greater depth and focus on a finer part of the action space compared to small subtrees. Where we note that the MO-HOO strategy roughly explores some parts of the actions space, whereafter it focusses on the region corresponding to the Pareto optimal area in the objective space. The third subplot depicts the sampled points in the objective space. The colour of the points indicates the iteration number, where red dots denote points obtained in later iterations. The MO-HOO samples the Pareto optimal region after just a few iterations. We also see that the MO-HOO strategy discovers a wide and diverse set of optimal solutions. This is a result of the optimistic term added to the estimates. Imagine the case where multiple estimates are optimal and therefore equally good. The algorithm then selects one and follows it down the tree until it arrives in an untouched leaf node. In the event that this action selection also retrieves an optimal reward vector, its estimate (plus the optimistic term) is propagated to the root node. Because the leaf is located deeper in the tree, this optimistic term will be less than for the other estimates that were originally present at the root node. Therefore, in the next iteration, the algorithm will not select the same action in the action space, but it will go for one of its original non-dominated estimates. As a result, a set of wide-spread and optimal solutions is retrieved.

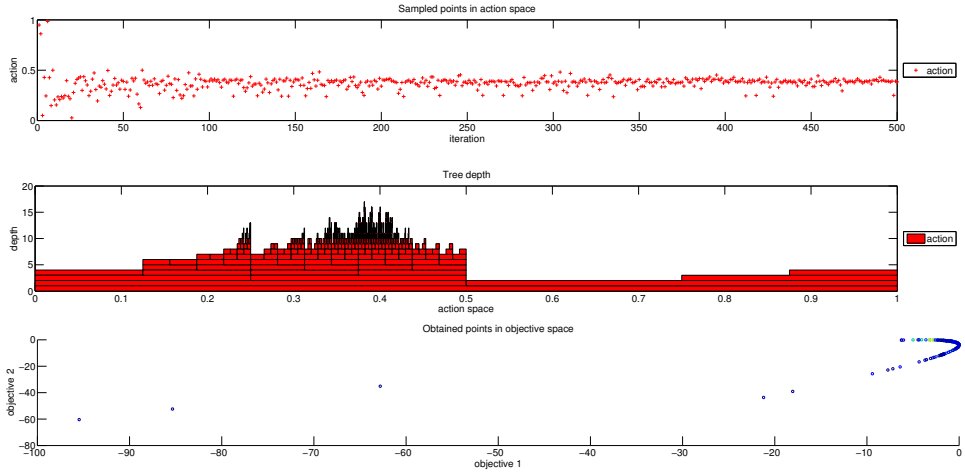


Figure 6.3: The MO-HOO strategy quickly retrieves the area in the action space that corresponds to the Pareto optimal area of the deterministic Schaffer 1 function (subplot 1 and 3). In subplot 2, we note that the algorithm first roughly explores the function at random and thereafter it focusses specifically and exclusively on the optimal part of the action space.

In Table 6.1, we present the average cardinality of all sampled policies during the learning process for each of the strategies. Recall that the cardinality indicators simply counts the number of Pareto non-dominated elements. Based on the results, we note MO-HOO acquires more than 6 times the amount of Pareto non-dominated solutions obtained by the random method. After 500 action selections, MO-HOO is able to retrieve on average 436.2 non-dominated points while the random strategy only obtained 69.5.

Strategy	Cardinality
Random	69.5
MO-HOO	**436.2**

Table 6.1: The cardinality indicator on the Schaffer 1 function for random exploration and MO-HOO. The goal is to maximise the number of Pareto optimal points.

6.2.5 Fonseca and Flemming function

The second problem is the Fonseca and Fleming function Fonseca and Fleming (1993):

$$\text{maximise } f(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}) = -(1 - e^{-\sum_{i=1}^{n}(\mathbf{x}_i - \frac{1}{\sqrt{n}})^2}) \\ f_2(\mathbf{x}) = -(1 - e^{-\sum_{i=1}^{n}(\mathbf{x}_i + \frac{1}{\sqrt{n}})^2}), \end{cases}$$

where $\boldsymbol{x}$ is a two-dimensional input vector with $\mathbf{x}_i \in [-4, 4]$ for input i. This bi-objective function contains a large and an entirely non-convex Pareto front which makes it especially appealing to assess an algorithm's ability to find a close and uniform approximation of the Pareto front. The function and its corresponding Pareto front can be found in Figure 6.4. While the previous optimisation function is entirely deterministic, we added Normal noise with $\sigma = 0.1$ to the values of each of the objectives.

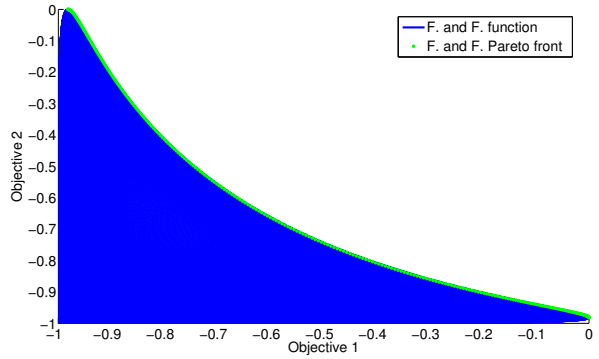


Figure 6.4: The objective space of the Fonseca and Flemming function. The Pareto front is highlighted by green dots.

In this environment the input vector itself is multi-dimensional. Therefore, we adjust the implementation of a node in the tree to take into account a multi-dimensional interval, i.e., one interval for each input variable. In this implementation, the tree splits the interval of the first variable for nodes located at uneven depth in the tree. In the case the node is at an even tree depth, the interval represented by the second variable is split.

In Figure 6.5, we depict the sampling performance of the MO-HOO strategy on this noisy function. As this function is significantly harder than the previous pay-off function, the probability of sampling the Pareto front increases more slowly than is the case for the Schaffer 1 function. When the learning process stopped after 1000 iterations, MO-HOO samples Pareto optimal actions in 88% of the cases. The naive random strategy reached a performance of only 20%.

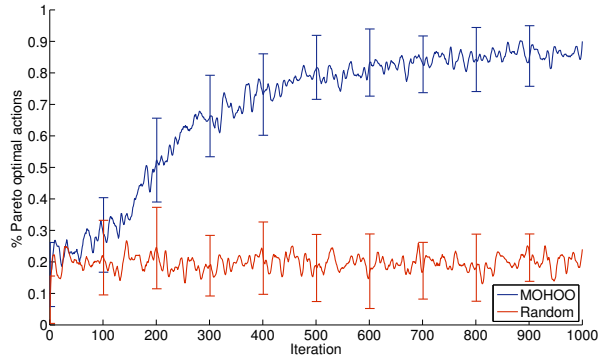


Figure 6.5: The sampling performance MO-HOO is compared to a naive random strategy. As this function is larger and noisy, the probability of the MO-HOO strategy sampling Pareto optimal solutions approaches 0.88 at the end of 1000 iterations.

In Figure 6.6, we investigate more closely the sampled actions in the action space, the development of the tree structure and the observed policies in the objective space. In the beginning of the process, we see the MO-HOO strategy rather randomly exploring the action space while focussing its samples on a particular part of the search space as time increases. After around 400 iterations of treating every part of the action space equally, a certain trend in the sampled policies starts to appear. However, this trend still seems to be rather noisy (subplot 1 of Figure 6.6). Also, compared to the more simple Schaffer 1 function, the optimal part of the tree does not stand out that much (subplot 2 of Figure 6.6). The main reason for this behaviour is probably the difficulty of the function together with the noise and the multi-dimensional input vector which influences the procedure the algorithm splits intervals.

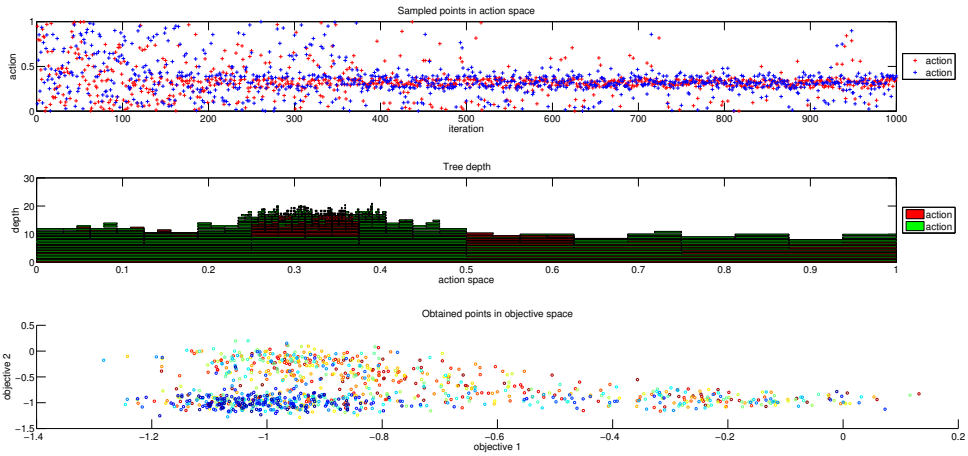


Figure 6.6: Subplots 1 and 2 depict that the MO-HOO strategy needs a bit more time on the Fonseca and Flemming function to explore the fruitful areas. We see that the algorithm needs around 400 iterations before it had clear idea on which parts to further explore.

The consequence of this practice is that the MO-HOO strategy did not entirely cover the continuous Pareto front as there are still some uncharted gaps. Table 6.2, we see that over 50 runs, MO-HOO obtained on average a set containing 16.22 solutions that offer an non-dominated balance between the two objectives. The random strategy obtained on average a set containing 8.1 non-dominated elements. However, as the results of Figure 6.6 denote, the non-dominated solutions of the MO-HOO strategy are much closer to the Pareto front and they clearly dominate the solutions obtained by the random strategy.

Strategy	Cardinality
Random	8.1
MO-HOO	**16.22**

Table 6.2: The cardinality indicator on the Fonseca and Flemming function for random exploration and MO-HOO.

6.2.6 Discussion

In this section, we have presented our first multi-policy algorithm for learning multiple policies in a simultaneous manner. We name this strategy the multi-objective hierarchical optimistic optimisation algorithm as it builds a tree of vectorial estimates that include

some statistics on the variance of the samples. This algorithm is particularly suited for real-life control problems that involve the optimisation of continuous parameters.

In the experimental section, we have seen that MO-HOO is an appropriate strategy to discover a set of non-dominated solutions at once and to sample accordingly. Although the pay-off functions in this setting only consider two objectives, the principles of the underlying tree-based algorithm can easily be extended to environments with more than two objectives. However, when increasing the number of objectives in the pay-off function, also the number of Pareto incomparable solutions increases. Therefore, the problem also gets harder for the optimisation algorithm and probably more running time is needed for the algorithm to discover the Pareto optimal regions.

To possibly improve the algorithm, one could investigate how to consider the $\boldsymbol{\mathcal{U}}$-vectors of non-leaf nodes in the backward computation to the root node. For each node, $\boldsymbol{\mathcal{U}}$-vectors store statistical information on the average quality of the subtree starting from that node. As this $\boldsymbol{\mathcal{U}}$-vector averages multiple reward estimates into a single estimate, we did not find it beneficial to be included in the backward computation. If it would suffice for the decision maker to sample the convex hull of the Pareto front, the $\boldsymbol{\mathcal{U}}$-vectors could provide supplementary guidance. Additionally the robustness of the MO-HOO strategy would also increase as the $\boldsymbol{\mathcal{U}}$-vectors provide average information over subtrees which is more resilient to noisy rewards.

So far, to the best of our knowledge, MO-HOO is the first multi-objective multi-policy $\mathcal{X}$-armed bandit solver. This has the advantage that it makes MO-HOO a very novel optimisation algorithm that still includes a lot of uncharted research questions. However, a rigorous experimental comparison of the algorithm is still needed. Due to time constraints our experimental evaluation is limited to a comparison with a naive random strategy. A possibility would be to perform an experimental study to genetic algorithms that also evolve a population of solutions, as presented in Section 3.2. However, these algorithms rarely consider the sampling cost and are often less suitable if the environment is stochastic.

In order to make MO-HOO an acceptable solution mechanism for control problems it would be of great importance to investigate its performance on some environments that are close to the real world. In Chapter 7, we take a first step and perform an initial experiment on a realistic simulation environment of the filling phase of a wet clutch.

6.3 Learning multiple policies in multi-state problems

In the previous section we have investigated a multi-policy MORL algorithm for single-state learning. In the case of a multi-state environment, the task at hand is a genuine sequential decision making problem. Over the years, a few multi-policy algorithms (Barrett and Narayanan, 2008; Lizotte et al., 2010, 2012) have been proposed that are all based on the same principle of the algorithm of White (1982). In that paper, a dynamic programming algorithm is proposed that computes a set of Pareto non-dominated policies. The dynamic

programming function is as follows:

$$\hat{Q}_{set}(s,a) = \mathbf{R}(s,a) \oplus \gamma \sum_{s' \in S} T(s'|s,a)\, V^{ND}(s') \tag{6.4}$$

where $\mathbf{R}(s,a)$ is the expected reward vector observed after taking action a in state s and $T(s'|s,a)$ is the corresponding transition probability of reaching state s' from (s,a). We refer to $V^{ND}(s')$ as the set of non-dominated vectors of the $\hat{Q}_{set}$'s of each action in s', as denoted in Eq. 6.5. The ND operator is a function that removes all Pareto dominated elements of the input set and returns the set of non-dominated elements.

$$V^{ND}(s') = ND(\cup_{a'} \hat{Q}_{set}(s',a')) \tag{6.5}$$

The $\oplus$ operator performs a vector-sum of a vector $\mathbf{v}$ and a set of vectors V. Summing two vectors can be performed simply by adding the corresponding components of the vectors.

$$\mathbf{v} \oplus V = \bigcup_{\mathbf{v}' \in V} (\mathbf{v} + \mathbf{v}') \tag{6.6}$$

The idea is that, after the discounted Pareto dominating rewards are propagated and the $\hat{Q}_{set}$'s converge to a set of Pareto dominating policies, the user can traverse the tree of $\hat{Q}_{set}$'s by applying a preference function. As highlighted in Section 2.3, a deterministic stationary policy suffices for single-objective reinforcement learning. In the case of MORL, White (1982) showed that deterministic non-stationary policies, i.e. policies that do not only condition on the current state but usually also on the timestep t, can Pareto dominate the best deterministic stationary policies. Although we omit the theoretical proof behind this theorem, the process is easily justified by considering the environment in Figure 6.7. In the example, adapted from Roijers et al. (2013), there is a single state S and three actions a_1, a_2 and a_3 that leave the state of the environment unchanged.

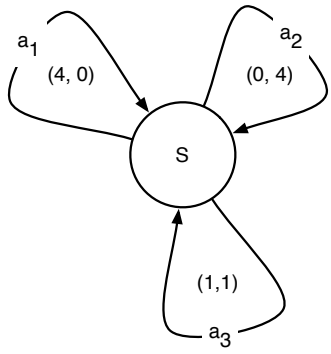


Figure 6.7: Deterministic non-stationary policies can Pareto dominate deterministic stationary policies.

The rewards of these actions are deterministic and yield $(4, 0)$, $(0, 4)$ and $(1, 1)$ for a_1, a_2 and a_3, respectively. If we consider only deterministic stationary policies, it is clear that there are three Pareto optimal policies π_1, π_2 and π_3 that always select action a_1, a_2 and a_3, respectively. The value functions of these policies are then

$$\mathbf{V}^{\pi_1}(S) = (\frac{4}{1-\gamma}, 0), \tag{6.7}$$

$$\mathbf{V}^{\pi_2}(S) = (0, \frac{4}{1-\gamma}), \tag{6.8}$$

$$\mathbf{V}^{\pi_3}(S) = (\frac{1}{1-\gamma}, \frac{1}{1-\gamma}), \tag{6.9}$$

where γ is the discount factor. Yet, if we would allow policies to be non-stationary, they are not constrained to selecting the same action at every occurrence of a particular state. For instance, we can construct a policy π_4 that alternates between selecting action a_1 and a_2. Since π_4 then conditions on the fact whether the timestep t is odd or even, the policy is said to be non-stationary. Consequently, the value function of π_4 equals

$$\mathbf{V}^{\pi_4}(S) = (\frac{4}{1-\gamma^2}, \frac{4\gamma}{1-\gamma^2}). \tag{6.10}$$

Accordingly, when $\gamma > \frac{1}{3}$, the non-stationary policy π_4 Pareto dominates the stationary policy π_3, confirming the statement in White (1982).

However, since many conditions based on the timestep t can exist, a lot of non-stationary policies similar to π_4 can be constructed that are Pareto non-dominated. Therefore, in infinite horizon problems with large values for the discount factor, the number of non-stationary policies the agent learns increases exponentially which can lead to a so-called explosion of the sets. In order to make the algorithm practically applicable, Wiering and de Jong (2007) propose the CON-MODP algorithm which solves the problem of non-stationary policies by introducing a consistency operator, but their work is limited to deterministic transition functions.

Our contribution in this section is Pareto Q-learning (PQL). To the best of our knowledge, PQL is the first temporal difference-based multi-policy MORL algorithm that does not use the linear scalarisation function. Thus, Pareto Q-learning is not limited to the convex hull, but it can learn the entire Pareto front, if enough exploration is provided, which is a standard condition for reinforcement learning algorithms. Pareto Q-learning also uses the principle of White (1982) as a starting point. As a result, PQL also learns deterministic non-stationary policies. For simplicity reasons, we currently only focus on episodic problems, i.e., environments with terminal states that end the episode. In Section 6.3.8, we analyse the challenges to extend PQL to ergodic environments.

Our PQL algorithm is particularly suited for on-line use, in other words, when the sampling cost of selecting appropriate actions is important and the performance should gradually increase over time. We also propose three evaluation mechanisms for the sets that provide a

basis for on-line action selection strategies in Section 6.3.3. These evaluation mechanisms use multi-objective indicators such as the hypervolume metric, the cardinality indicator and the Pareto dominance relation in order to select the best possible actions throughout the learning process. In Section 6.3.6, the Pareto Q-learning algorithm is evaluated on multiple environments with two and three objectives and its performance is compared w.r.t. several MORL algorithms. First, we elaborate on how we extend the core principles of the algorithm of White (1982) in Section 6.3.1. More precisely, we present a novel mechanism that allows to incrementally learn and update sets of non-dominated vectors over time.

6.3.1 Set-based Bootstrapping

The single-objective Q-learning bootstrapping rule updates an estimate of an (s, a)-pair based on the reward and an estimate of the next state (Watkins, 1989). The update rule guarantees that the $\hat{Q}$-values converge to their expected future discounted reward, even when the environment's transition function is stochastic. In this section, we analyse the problem of bootstrapping sets of vectors. We first present a naive approach whereupon we present our novel Pareto Q-learning algorithm.

Naive approach

The set-based bootstrapping problem boils down to the general problem of updating the set of vectors of the current state-action (s, a)-pair with an observed reward vector $\mathbf{r}$ and a set of non-dominated vectors of the next state, $ND(\cup_{a'}\hat{Q}_{set}(s', a'))$ over time. The difficulty in this process arises from the lack of *correspondence* between the vectors in the two sets, i.e., it is not clear which vector of the set of the current (s, a)-pair to update with which vector in s'. This correspondence is needed to perform a pairwise update of each vector in $\hat{Q}_{set}(s, a)$ with the corresponding vector (if any) in the other set, as denoted in Figure 6.8.

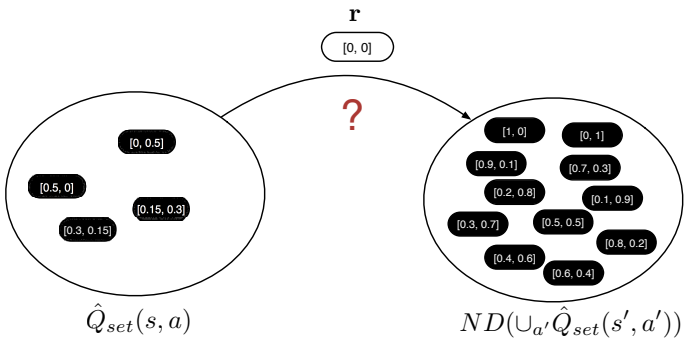


Figure 6.8: Set-based bootstrapping: the problem of updating over time the set of vectors of the current state-action pair with the observed reward vector and the optimal vectors of the next state. There is no explicit correspondence between the elements in both sets, so as to perform a pairwise update.

A possible solution is to make this correspondence explicit by labelling or tagging the vectors in the two sets. When vectors in the sets of (s, a) and s' are tagged with the same label or colour, the bootstrapping process knows that these vectors can be updated in a pairwise manner. More precisely, the process would be as follows: when sampling an action a in s for the first time, the vectors in the set of the next state $ND(\cup_{a'}\hat{Q}_{set}(s', a'))$ are labeled with a unique tag. Next, the bootstrapping process can continue for each vector in s' individually and the tag is copied to the set of (s, a). This process is illustrated in Figure 6.9 (a) for a bi-objective environment. Subsequently, when action a is sampled in future timesteps, we would possess an actual correspondence between the vectors in the two sets and we can perform a pairwise update for each objective of each vector with the same label (Figure 6.9 (b)). However, the main problem with this naive solution is that these sets are not *stable* but can change over time. We highlight two main cases that can occur in a temporal-difference setting:

- It is possible that the set of (s, a) is updated with vectors from s' at timestep t, while actions in s' that are previously unexplored are sampled at timestep $t + 1$. Then, possibly new non-dominated vectors may appear in $ND(\cup_{a'}\hat{Q}_{set}(s', a'))$. When, in future episodes, the set of (s, a) is to be updated again, there are elements in s' that are not bootstrapped before and the correspondence between the sets is incomplete (Figure 6.9 (c)).
- As estimates are being updated over time, it is very likely that vectors in s' that were non-dominated at timestep t may become dominated by other vectors at timestep $t + 1$. In Figure 6.9 (d), we see that in that case the correspondence no longer holds, i.e., different labels appear in the two sets. As a consequence, learning would have to begin from scratch again for those vectors. Especially in early learning cycles, the vectorial estimates can repeatedly switch between being non-dominated and dominated. Hence, this naive updating process would waste a lot of samples before the vectors mature.

It is clear that such a naive updating procedure would become even more cumbersome and complex in environments with stochastic transitions. As a result, it would not be generally applicable to a wide range of problem domains.

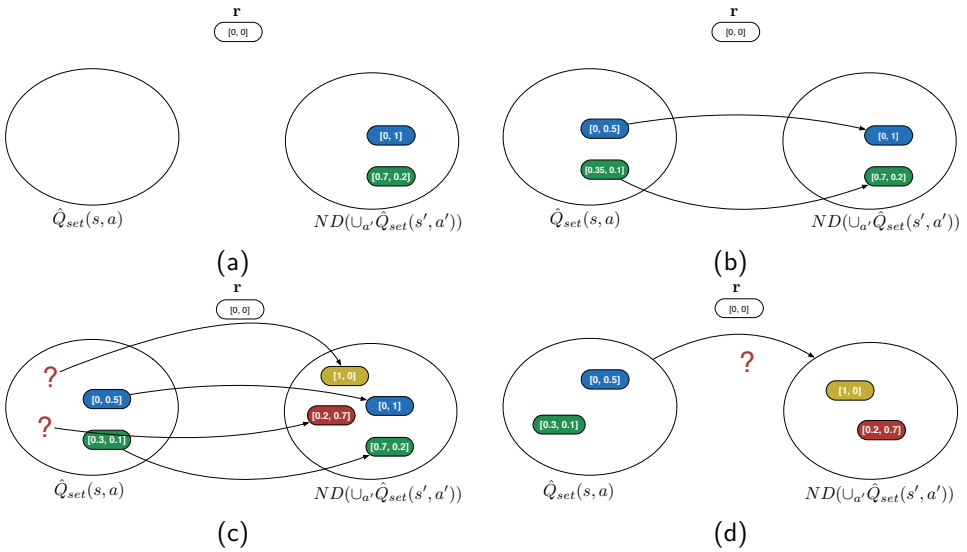


Figure 6.9: Several situations can occur when updating a set with another set over time. In (a), we would naively label the vectors of s' with a certain colour when sampling action a in s for the first time. In (b), we note that the labeled and coloured vectors of s' are now bootstrapped and present in (s, a). As the colours are also copied in (s, a), the correspondence between the vectors in (s, a) and s' is explicit and in future timesteps the vectors can be updated in a pairwise manner. Illustrations (c) and (d) highlight the different situations one should account for as the sets are not stable but can change over time. For instance, new vectors can appear in s' (c) or estimates that were non-dominated can become dominated (d). We refer to Section 6.3.1 for more details.

Our approach: Learning immediate and future reward separately

In the presentation of our updating principle, we first limit ourselves to environments with deterministic transition functions. We then proceed to highlight the minimal extensions to the algorithm to also cover stochastic transitions.

In standard, single-objective Q-learning, $\hat{Q}$-values store the sum of the estimated value of the immediate reward and the future discounted reward as presented in Section 2.4.2. Our idea consists of storing this information separately. We use $\overline{\mathfrak{R}}(s, a)$ to denote the average observed immediate reward vector of (s, a) and $ND_t(s, a)$ to represent the set of non-dominated vectors in the next state of s that is reached through action a at timestep t. The next state of s is determined by observing the transitions during learning. By storing $\overline{\mathfrak{R}}(s, a)$ and $ND_t(s, a)$ separately, we allow them to converge separately as well. This way, no explicit correspondence between the two sets is required and the current set of non-dominating policies at timestep t, $ND_t(s, a)$ is allowed to evolve over time. The $\hat{Q}_{set}$ of (s, a) can be calculated *at run time* by performing a vector-sum over the average

immediate reward vector and the set of discounted Pareto dominating future rewards:

$$\hat{Q}_{set}(s,a) \leftarrow \overline{\mathfrak{R}}(s,a) \oplus \gamma ND_t(s,a) \tag{6.11}$$

Whenever the action a in s is selected, the average immediate reward vector $\overline{\mathfrak{R}}(s,a)$ is updated and the $ND_t(s,a)$ list is updated using the non-dominated $\hat{\mathbf{Q}}$-vectors in the $\hat{Q}_{set}$ of every action a' in s', i.e., $ND(\cup_{a'}\hat{Q}_{set}(s',a'))$.

We present an algorithmic outline of the Pareto Q-learning algorithm in Algorithm 7. The algorithm starts by initialising the $\hat{Q}_{set}$'s as empty sets. In each episode, an action is selected using a particular action selection strategy (line 5). How we actually perform the action selection based on the $\hat{Q}_{set}$'s will be presented in the subsequent section. Afterwards, the environment transfers the agent to state s' and provides the reward vector $\mathbf{r}$. In state s', the non-dominated $\hat{\mathbf{Q}}$-vectors for each action are retrieved at line 8 and are discounted. At line 9, the average immediate reward for each objective, $\overline{\mathfrak{R}}(s,a)$, is iteratively updated given the new reward $\mathbf{r}$ and the number of times that action a is sampled, denoted by $n(s,a)$. The algorithm proceeds until the $\hat{Q}_{set}$'s converge or after a predefined number of episodes.

Algorithm 7 Pareto Q-learning algorithm

1: Initialise $\hat{Q}_{set}(s,a)$'s as empty sets
2: **for** each episode t **do**
3: Initialise state s
4: **repeat**
5: Choose action a from s using a policy derived from the $\hat{Q}_{set}$'s
6: Take action a and observe state $s' \in S$ and reward vector $\mathbf{r} \in \mathbb{R}^m$
7:
8: $ND_t(s,a) \leftarrow ND(\cup_{a'}\hat{Q}_{set}(s',a'))$ ▷ Update ND policies of s' in s
9: $\overline{\mathfrak{R}}(s,a) \leftarrow \overline{\mathfrak{R}}(s,a) + \frac{\mathbf{r}-\overline{\mathfrak{R}}(s,a)}{n(s,a)}$ ▷ Update average immediate rewards
10: $s \leftarrow s'$ ▷ Proceed to next state
11: **until** s is terminal
12: **end for**

The updating principle can also be extended to stochastic environments, where the transition probability $T(s'|s,a) \neq 1$ for some next state s', given state s and action a. In the case of stochastic transition functions, we store the expected immediate and future non-dominated rewards per (s,a,s')-tuple that is observed during sampling, i.e., $\overline{\mathfrak{R}}(s,a,s')$ and $ND_t(s,a,s')$, respectively. By also considering the observed frequencies of the occurrence of next state s' per (s,a)-pair, i.e., $F^{s'}_{s,a}$, we estimate $T(s'|s,a)$ for each (s,a). Hence, we learn a small model of the transition probabilities in the environment, similar to Dyna-Q (Sutton and Barto, 1998), which we use to calculate a weighted pairwise combination between the sets. To combine a vector from one set with a vector from the other set, we propose the $\mathcal{C}$-operator, which simply weighs them according to the observed

transition frequencies:

$$\mathcal{C}(\hat{\mathbf{Q}}(s,a,s'),\hat{\mathbf{Q}}(s,a,s'')) = \frac{F_{s,a}^{s'}}{\sum_{s'''\in S} F_{s,a}^{s'''}}\hat{\mathbf{Q}}(s,a,s') + \frac{F_{s,a}^{s''}}{\sum_{s'''\in S} F_{s,a}^{s'''}}\hat{\mathbf{Q}}(s,a,s'') \quad (6.12)$$

6.3.2 Notions on convergence

In this section, we sketch a convergence proof of the Pareto Q-learning algorithm. A sketch of the convergence of the algorithm can be accomplished by imitating the process of the proof of the single-objective Q-learning algorithm. This algorithm is a recursive algorithm that, in each stage, updates its estimates based on newly acquired information of the environment. Since the correction terms of each update are random, Tsitsiklis (1994) combines ideas from stochastic approximation and convergence theory of parallel asynchronous algorithms to prove the convergence of Q-learning. In essence, Tsitsiklis (1994) argues the update rule of the Q-learning algorithm in Equation 2.8 is a contraction. Let us say that this update rule is a function $\mathcal{G}$ which takes as input a vector $q \in \mathcal{R}^n$, then $\mathcal{G}(\mathcal{G}(\mathcal{G}(\mathcal{G}(...\mathcal{G}(x)))))$ converges to a fixed-point for all $x \in \mathcal{R}^n$. This means that regardless of the initialisation of the Q-values, the update rule makes these values converge to their optimal Q^*-values under acceptable conditions. Basically, $\mathcal{G}$ is a contraction mapping w.r.t. maximum norm $||\ ||_\infty$. This means that the more times $\mathcal{G}$ is applied on a Q-value, the closer the estimate will become to its optimal value Q^*.

In the case of Pareto Q-learning, the situation is more complex because of two reasons. Firstly, we learn Q-vectors instead of Q-values. Secondly, and more importantly, we learn a set consisting of multiple vectors. Therefore, a traditional norm such as $||\ ||_\infty$ is not applicable in this context. Luckily, the Hausdorff distance can be used to measure the distance between sets of vectors. The Hausdorff distance between two sets X and Y of discrete elements can be calculated as follows:

$$d_H(X,Y) = \max\{\min_{\forall x\in X,\forall y\in Y} d(x,y), \min_{\forall y\in Y,\forall x\in X} d(y,x)\}. \quad (6.13)$$

Basically, the metric simply calculates the longest shortest distance between elements of the two sets.

Although we do not provide a formal theoretical proof on the convergence of PQL, its convergence can be argued by the following experiment. In Figure 6.10, we calculate at each timestep the Hausdorff distance between the learned set and the elements of the Pareto front. At first we note that the initial learned set only contains a null element and the distance to the optimal set is large. As the Pareto Q-learning update rule is applied more and more, the distance between the sets gradually decreases. Eventually, the distance becomes zero, meaning that the learned sets becomes identical to the Pareto front and all the optimal policies are learned.

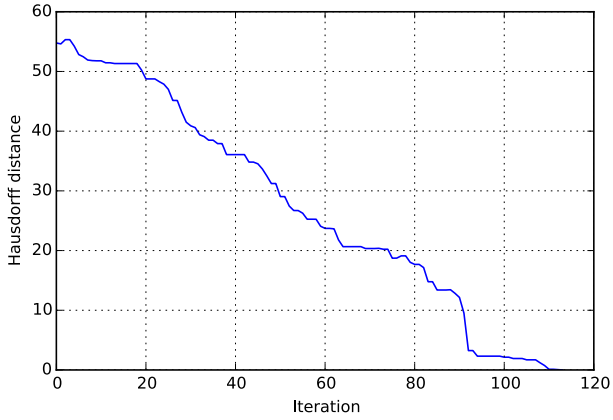


Figure 6.10: The Hausdorff distance between the learned set to the Pareto front over time. At first, the distance is large but the update rule of Pareto Q-learning decreases the distance until it becomes zero. Hence, eventually, the learned set becomes equal to the Pareto front and all the optimal policies are learned.

6.3.3 Set evaluation mechanisms

In reinforcement learning, the on-line performance is crucial. Therefore, it is interesting to see how the standard exploration strategies, such as ϵ-greedy, can be applied on the $\hat{Q}_{set}$'s during learning. In this section, we propose three evaluation mechanisms that obtain a scalar indication of the quality of a $\hat{Q}_{set}$. These scalar evaluations are used in action selection strategies to balance exploration and exploitation. We name these techniques *set evaluation* mechanisms.

Hypervolume set evaluation

The first set evaluation mechanism we propose uses the hypervolume measure of Section 4.4.1 to evaluate the $\hat{Q}_{set}$'s. Recall that the hypervolume indicator is well-suited for two reasons: (1) it is the only quality indicator to be strictly monotonic with the Pareto dominance relation and (2) it provides a scalar measure of the quality of a set of vectors. An outline of the evaluation mechanism is given in Algorithm 8. First, we initialise the list where the evaluations of each action of s will be stored. At line 4, we calculate the $\hat{Q}_{set}$ for each action and we compute its hypervolume which we append to the list. The list of evaluations can then be used in an action selection strategy, similar to the single-objective case. For instance, when selecting an action greedily, the action corresponding to the $\hat{Q}_{set}$ with the largest hypervolume is selected. When the $\hat{Q}_{set}$'s are empty, the hypervolume

of each action is 0 and an action is selected uniformly at random.[1] This set evaluation mechanism, in combination with Pareto Q-learning, is referred to as *HV-PQL*.

Algorithm 8 Hypervolume Q_{set} evaluation

1: Retrieve current state s
2: $\text{evaluations} = \{\}$
3: **for** each action a **do**
4: $\quad hv_a \leftarrow HV(\hat{Q}_{set}(s,a))$
5: $\quad$ Append hv_a to evaluations ▷ Store hypervolume of the $\hat{Q}_{set}(s,a)$
6: **end for**
7: **return** evaluations

Cardinality set evaluation

An alternative to the previous evaluation mechanism uses the cardinality quality indicator to evaluate actions. This evaluation mechanism, named *C-PQL*, considers the number of Pareto non-dominated $\hat{\mathbf{Q}}$-vectors of the $\hat{Q}_{set}$ of each action.

The rationale behind this evaluation mechanism is that it can heuristically guide the search process by providing a degree of domination one action has over other actions, locally in a state. It is expected that these actions then have a larger probability to lead to global Pareto optimal solutions. Especially when estimates are not yet mature, it might be interesting to bias the action selection to actions with a large number of non-dominated solutions. An outline of the evaluation mechanism is given in Algorithm 9. At line 2, we initialise a list where we store the individual $\hat{\mathbf{Q}}$-vectors of the $\hat{Q}_{set}$, together with a reference to its corresponding action a (line 5). At line 8, we remove all dominated $\hat{\mathbf{Q}}$-vectors using the ND operator, such that only the non-dominated $\hat{\mathbf{Q}}$-vectors remain in the $NDQs$ list. Using this list, the underlying action selection strategy can simply count the number of times each action a of s remains in the list of Pareto dominating $\hat{\mathbf{Q}}$-vectors, i.e., the $NDQs$ list, and eventually perform the action selection. Thus, when selecting an action greedily, the action that relates to the largest number of Pareto dominating $\hat{\mathbf{Q}}$-vectors over all actions in s is selected.

Pareto set evaluation

The third evaluation mechanism is a simplified version of the cardinality metric. Instead of considering the number of non-dominated elements in the $\hat{Q}_{set}$ of each action in s, we simply consider whether or not action a contains a non-dominated vector. The approach eliminates any actions which are dominated, and then randomly selects amongst the non-dominated actions. Hence, this mechanism only relies on the Pareto relation and is therefore called *PO-PQL*. PO-PQL removes the bias that the cardinality indicator in

[1] This is also the case for the other set evaluation mechanisms presented in this dissertation.

Algorithm 9 Cardinality Q_{set} evaluation

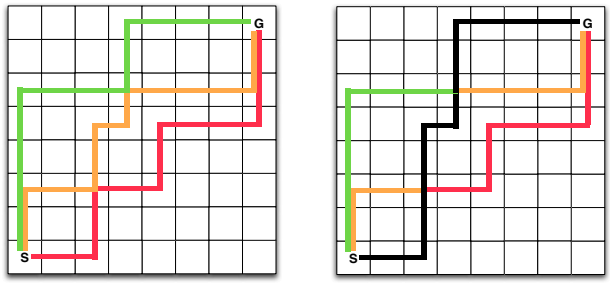
```
Retrieve current state s
allQs = {}
for each action a in s do
    for each Q̂ in Q̂_set(s, a) do
        Append [a, Q̂] to allQs          ▷ Store for each Q̂-vector a reference to a
    end for
end for
NDQs ← ND(allQs)                        ▷ Keep only the non-dominating solutions
return NDQs
```

C-PQL might have for actions with a large number of non-dominated vectors over actions with just a few. The rationale behind this mechanism is to have a more relaxed evaluation of the actions and to treat every non-dominated solution equally.

6.3.4 Consistently tracking a policy

The set evaluation mechanisms in Section 6.3.3 provide the necessary tools to perform action selection during learning, i.e., balancing the exploration towards uncharted areas of the state space and the exploitation of non-dominated actions. However, at any moment in time, it might be necessary to apply the learned policies. In single-objective reinforcement learning, the learned policy can be easily tracked by applying the $\arg\max$-operator over all actions in each state, i.e., applying greedy action selection. In the case of a multi-policy problem, we are learning multiple policies at the same time which requires an adapted definition of a greedy policy in MORL.

Because of the nature of multi-policy setting, one needs to select actions *consistently* in order to retrieve a desired policy based on the $\hat{\mathbf{Q}}$-vectors. If one would select actions based on *local* information about the 'local' Pareto front attainable from each action, then there is no guarantee that the cumulative reward vectors obtained throughout the episode will be *globally* Pareto optimal. This process is highlighted in Figure 6.11 (a) where the state space is an 8×8 grid and three global Pareto optimal policies exist, each given a different colour. In Figure 6.11 (b), we select actions that are locally non-dominated, i.e, non-dominated within the current state. The black policy is a sequence of locally optimal actions, as it always overlaps with one of the coloured lines, however, the resulting policy is not globally Pareto optimal. To conclude, when in a state where multiple actions are considered non-dominated, and thus incomparable, one can not randomly select between these actions when exploiting a chosen balance between criteria. Instead, actions need to be selected consistently.

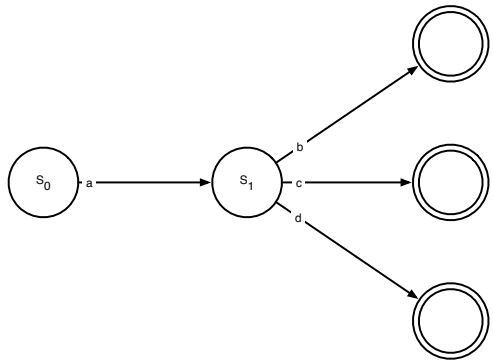


(a) (b)

Figure 6.11: (a) In this environment, there is a green, yellow and red Pareto optimal action sequence that is globally optimal. (b) Selecting actions that are locally non-dominated within the current state does not guarantee that the entire policy is globally Pareto optimal. Hence, the information about the global Pareto front has been lost in the local Pareto front.

Globally greedy policy

In order to solve the problem of locally optimal actions that are globally dominated, we define a greedy policy as a policy π that consistently *follows* or *tracks* a given expected return vector $\mathbf{V}^{\pi}(s)$ from a state s so that its return equals $\mathbf{V}^{\pi}(s)$ in expectation. Therefore, we need to retrieve π, i.e., which actions to select from a start state to a terminal state. However, due to the stochastic behaviour of the environment, it is not trivial to select the necessary actions so as to track the desired return vectors. Let us consider the small bi-objective MDP with deterministic transitions in Figure 6.12.



Action	$\overline{\mathfrak{R}}(s,a)$	$ND_t(s,a)$	$\hat{Q}_{set}(s,a)$
a	(0.2, 0.0)	((0.9, 0.5), (2.0, 0.4), (0.0, 0.6))	((1.1, 0.5), (2.2, 0.4), (0.2, 0.6))
b	(0.9, 0.5)	()	(0.9, 0.5)
c	(2.0, 0.4)	()	(2.0, 0.4)
d	(0.0, 0.6)	()	(0.0, 0.6)

Figure 6.12: A small multi-objective MDP and its corresponding Q_{set}'s. As we store the expected immediate and future non-dominated vectors separately, we can consistently follow expected return vectors from start to end state.

When the agent reaches a terminal state, denoted by a double circle, the episode is finished. Assume that the discount factor γ is set to 1 for simplicity reasons. Once the Q_{set}'s have converged separately, we can identify three Pareto optimal policies in the start state s_0. These policies have corresponding expected reward vectors $(1.1, 0.5)$, $(2.2, 0.4)$ and $(0.2, 0.6)$. When one is for instance interested in following the policy with an expected reward vector of $(2.2, 0.4)$, the agent should select action a as the vector $(2.2, 0.4)$ is an element of the $\hat{Q}_{set}$ of action a (and there is no other option). But, once in the next state, the next action to select is not clear when one only stores the converged Q_{set}'s. Hence, should the agent select action b, c or d to acquire a return of $(2.2, 0.4)$ at the end of the episode? The approach we propose to solve this issue is based on the separation of the average immediate and future rewards, i.e., we can simply subtract the average immediate reward from the expected return we are targeting, in order to retrieve the next action to select. This way, we can consistently follow the expected return from state s, $\mathbf{V}^{\pi}(s)$, throughout the entire state space. In the example, the agent should select the action that contains $(2.2, 0.4) - (0.2, 0) = (2.0, 0.4)$ in its Q_{set}, i.e., action c. The pseudo-code of the tracking algorithm for environments with deterministic transitions is listed in Algorithm 10. The agent starts in a starting state s of the environment and has to follow a particular policy so as to obtain the expected value of the policy from that state, i.e., $\mathbf{V}^{\pi}(s)$, at the end of the episode. For each action of the action set A, we retrieve both the averaged immediate reward $\overline{\mathfrak{R}}(s,a)$ and $ND_t(s,a)$, which we discount. If the sum of these two components equals the target vector to follow, we select the corresponding action and

proceed to the next state. The return $target$ to follow in the next state s' is then assigned to $\mathbf{ND}$ and the process continues until a terminal state is reached.

Algorithm 10 Track policy π given the expected reward vector $\mathbf{V}^{\pi}(s)$ from state s

1: $target \leftarrow \mathbf{V}^{\pi}(s)$
2: **repeat**
3: **for** each a in A **do**
4: Retrieve $\overline{\mathfrak{R}}(s,a)$
5: Retrieve $ND_t(s,a)$
6: **for** each $\mathbf{ND}$ in $ND_t(s,a)$ **do**
7: **if** $\gamma\,\mathbf{ND} + \overline{\mathfrak{R}}(s,a) = target$ **then**
8: $s \leftarrow s' : T(s'|s,a) = 1$
9: $target \leftarrow \mathbf{ND}$
10: **end if**
11: **end for**
12: **end for**
13: **until** s is not terminal

When the transition scheme is stochastic, the tracking algorithm is adapted at certain positions. The pseudo-code of the algorithm can be found in Algorithm 11. Once we have found the action belonging to a certain target at line 6, we simply select the action and arrive in state s' given $T(s'|s,a)$. In the next phase of the algorithm, we decide on the next target to follow. We decompose the original $\hat{\mathbf{Q}}(s,a)$ into its individual components, which are the separate $\hat{\mathbf{Q}}(s,a,s'')$-vectors for the observed (s,a,s'')-triplets. At line 9, we select the vector for the current transition to state s', i.e., $\hat{\mathbf{Q}}(s,a,s')$. Similar to the algorithm for deterministic transitions, we decompose that vector into its average immediate and its future discounted rewards. The latter one then becomes the new target to follow. Additionally, when the vectors have not entirely converged yet, the equality operator at line 6 should be relaxed. In this case, the action is to be selected that minimises the difference between the left and the right term. In our experiments, we select the action that minimises the Manhattan distance between these terms.

Algorithm 11 Track policy π given the expected reward vector $\mathbf{V}^{\pi}(s)$ from state s

$target \leftarrow \mathbf{V}^{\pi}(s)$
repeat
  for each a in A **do**
    Retrieve $\hat{Q}_{set}(s,a)$
    for each $\mathbf{Q}$ in $\hat{Q}_{set}(s,a)$ **do**
      if $\mathbf{Q} = target$ **then**
        $s \leftarrow s' : T(s'|s,a) \neq 1$
        Knowing that $\mathbf{Q} = \mathcal{C}(\hat{\mathbf{Q}}(s,a,s'),\ldots)$ using Equation 6.12
        and that $\hat{\mathbf{Q}}(s,a,s') = \overline{\mathfrak{R}}(s,a,s') + \gamma\,\mathbf{ND}$
        Retrieve $\overline{\mathfrak{R}}(s,a,s')$
        Retrieve $\mathbf{ND}$
        $target \leftarrow \mathbf{ND}$
      end if
    end for
  end for
until s is not terminal

6.3.5 Performance assessment of multi-policy algorithms

Before we analyse the experiments, we first discuss the general challenges in assessing the performance of on-line multi-policy MORL algorithms. In single-objective reinforcement learning, an algorithm is usually evaluated by its average reward accumulated over time. The curve of the graph then indicates both the speed of learning and the final performance of the converged policy. In multi-objective reinforcement learning, the performance assessment is more complex because of two main reasons: (1) the reward signal is vectorial and not scalar and (2) there exists no total ordering of the policies but there is a set of *incomparable* optimal policies.

For scalarised MORL algorithms that converge to a single policy, Vamplew et al. (2010) propose to employ the hypervolume indicator on the *approximation* set of policies, i.e., the policies that are obtained after applying a greedy policy for a range of experiments with varying parameters in the scalarisation functions. The hypervolume of the approximation set can then be compared to the hypervolume of the *true* Pareto front, i.e., the set of Pareto optimal policies of the environment. Each experiment then represents an individual run of a scalarised MORL algorithm with a specific weight vector $\mathbf{w}$. In the case of tracking globally greedy policies, we can adopt the mechanism by Vamplew et al. (2010) and calculate the hypervolume of the cumulative reward vectors obtained by the tracking algorithm of Section 6.3.4 for each of the non-dominated vectors in the $ND(\cup_a \hat{Q}_{set}(s_0,a))$, where s_0 is the start state. The hypervolume of each of these vectors should then approach the hypervolume of the Pareto front. It is important to note that in this way, we will evaluate and track as many policies as there exist non-dominated $\hat{\mathbf{Q}}$-vectors in the start state for all actions.

6.3.6 Benchmarking Pareto Q-learning

In this section, we analyse the performance of the Pareto Q-learning algorithm on two test problems for each of the three set evaluation mechanisms, i.e., with either the hypervolume, cardinality or Pareto evaluation mechanism. The algorithms are tested on several benchmark environments with a linear, convex and non-convex Pareto front. All the experiments are averaged over 50 runs and their 95% confidence interval is depicted at regular intervals.

The Pyramid MDP

The Pyramid MDP is a new and simple multi-objective benchmark environment. A visual representation of the world is depicted in Figure 6.13. The agent starts in the down-left position, denoted by a black dot at $(0, 0)$, and it can choose any of the four cardinal directions (up, down, left and right). The transition function is stochastic so that with a probability of 0.95 the selected action is performed and with a probability of 0.05 a random transition is executed to a neighbouring state. The red dots represent terminal states. The reward scheme is bi-objective and returns a reward drawn from a Normal distribution with $\mu = -1$ and $\sigma = 0.01$ for both objectives, unless a terminal state is reached. In that case, the x and y position of the terminal state is returned for the first and second objective, respectively. The Pareto front is therefore linear as depicted in Figure 6.14.

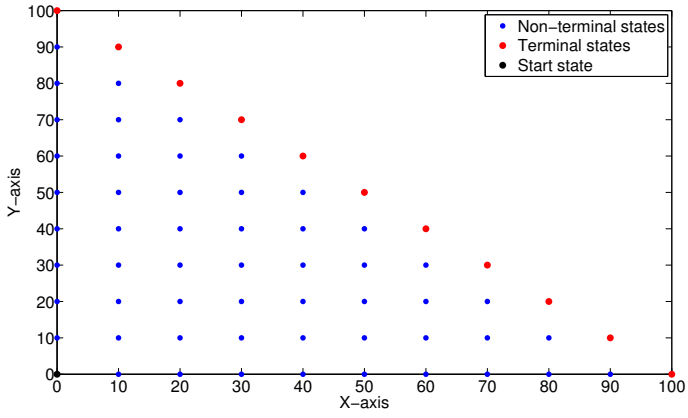


Figure 6.13: The Pyramid MDP: the agent starts in the down-left position and can select actions until a terminal state is reached, denoted by a red dot.

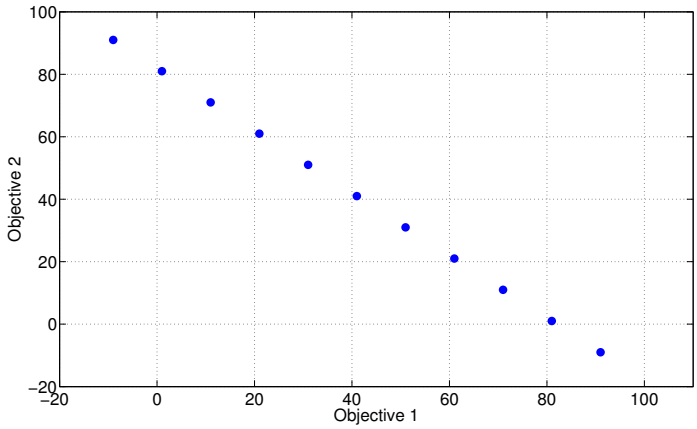


Figure 6.14: The corresponding linear Pareto front of the Pyramid MDP.

As we are learning multiple policies simultaneously, which potentially may involve different parts of the state space, we found it beneficial to employ a train and test setting, where in the train mode, we learn with an ϵ-greedy action selection strategy with decreasing epsilon.[2] In the test mode of the algorithm, we perform multiple greedy policies using Algorithm 10 for every element in $ND(\cup_a \hat{Q}_{set}(s_0, a))$ of the start state s_0 and we average the accumulated returns along the paths. Each iteration, these average returns are collected and the hypervolume is calculated.

In Figures 6.15 and 6.16, we present the results of learning and sampling Pareto optimal policies in the Pyramid MDP environment for the train and test phases, respectively.

[2]At episode eps, we assigned ϵ to be 0.997^{eps} to allow for significant amounts of exploration in early runs while maximising exploitation in later runs of the experiment.

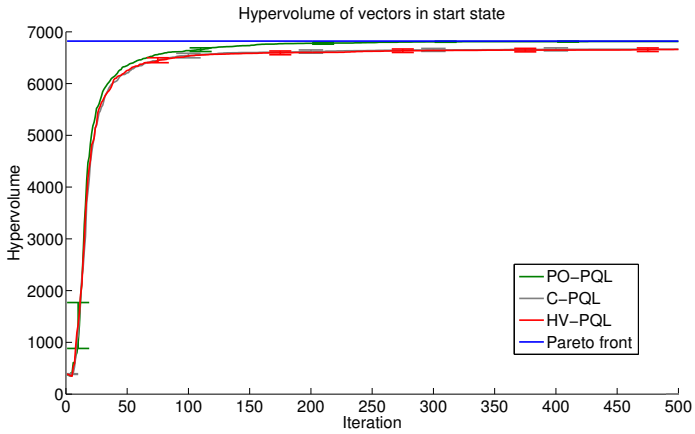


Figure 6.15: We depict the hypervolume of the estimates in the start state of the stochastic Pyramid MDP and we note that the entire Pareto front is learned very quickly during the learning phase.

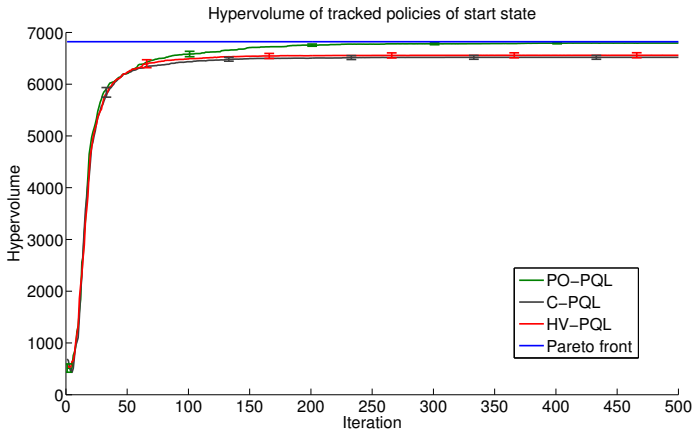


Figure 6.16: We track the estimates learned in Figure 6.15 through the state space and denote that the hypervolume of their average returns approaches the Pareto front as well.

In Figure 6.15, we depict the hypervolume over time of the estimates in the start state s_0.[3] Hence, for each iteration, we calculate $HV(\cup_a \hat{Q}_{set}(s_0, a))$. We see that each of the set evaluation mechanisms guide the Pareto Q-learning algorithm very well as the hypervolume of the learned estimates approaches the hypervolume of the Pareto front (γ

[3]In the Pyramid MDP, the reference point for the hypervolume calculation in both *HV-PQL* and the performance assessment is specified to $(-20, -20)$ after observing the reward scheme.

is set to 1). Based on the graph, we see that each of the set evaluation mechanisms has very similar performance in early stages of the learning process. After around hundred iterations, however, we note a small distinction in performance between C-PQL and HV-PQL on the one hand and PO-PQL on the other hand. Closer investigation of the results taught us that this difference is caused by the fact that, once the estimates become stable, C-PQL and HV-PQL still create a total order out of the set of Pareto optimal estimates, even though they are incomparable. That is why, in later iterations of the learning phase, C-PQL and HV-PQL provide a (too) large bias towards particular areas of the state and action space and therefore some estimates are no longer updated. Hence, the very close, but not coinciding curves of their learning graphs. PO-PQL does not provide a total order, but keeps the partial order that the multi-objective nature of the problem entails. Therefore, it treats every Pareto optimal solution equally and the estimates are updated much more consistently.

In Figure 6.16, we depict the results of the tracking algorithm of Section 6.3.4 that globally follows every element of the start state in Figure 6.15 (a). We see that the hypervolume of the average returns of each the estimated $\hat{\mathbf{Q}}$-vectors in $ND(\cup_a \hat{Q}_{set}(s,a))$ is very similar to the learned estimates themselves. We see that tracking the estimates obtained by PO-PQL allows to sample the entire Pareto front over time.

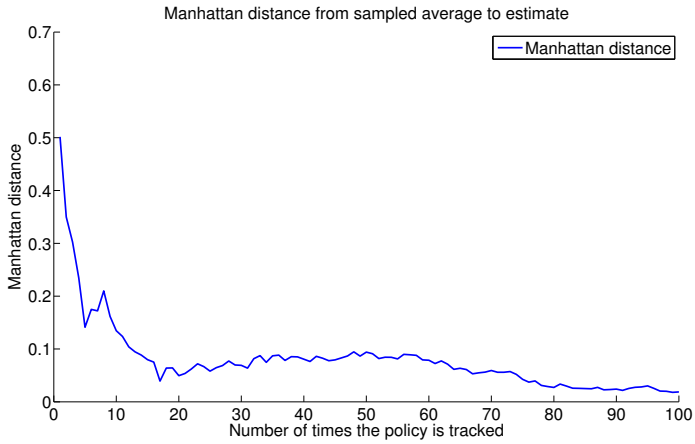


Figure 6.17: The performance of the tracking algorithm for a specific estimate. We see that the Manhattan distance of the running average of the return vector approaches the tracked estimate over time.

In Figure 6.17, we see the tracking algorithm at work to retrieve the policy of a specific vectorial estimate from the start state. In the figure, we denote the Manhattan distance of the running average return to the estimate after learning. We see that averaging the return of the policy obtained by the tracking algorithm over time approaches the estimate predicted at the start state, i.e., the distance becomes zero in the limit.

The Pressurised Bountiful Sea Treasure environment

In order to evaluate the Pareto Q-learning algorithm on an environment with a larger number of objectives, we analyse its performance on the Pressurised Bountiful Sea Treasure (PBST) environment of Section 5.2.5. This environment is very similar to the standard Deep Sea Treasure (DST) environment (Vamplew et al., 2010) with the slight modification of an additional objective.

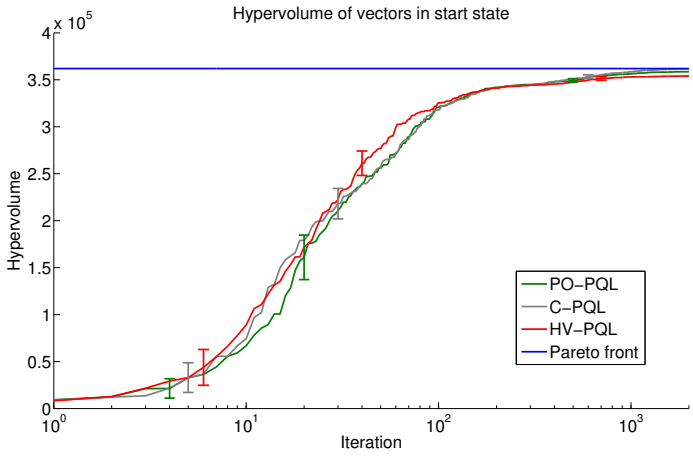


Figure 6.18: The results on the PBST environment. We depict the hypervolume over time of the learned estimates in the start state. As time progresses, the vectors in the start state approach their true values.

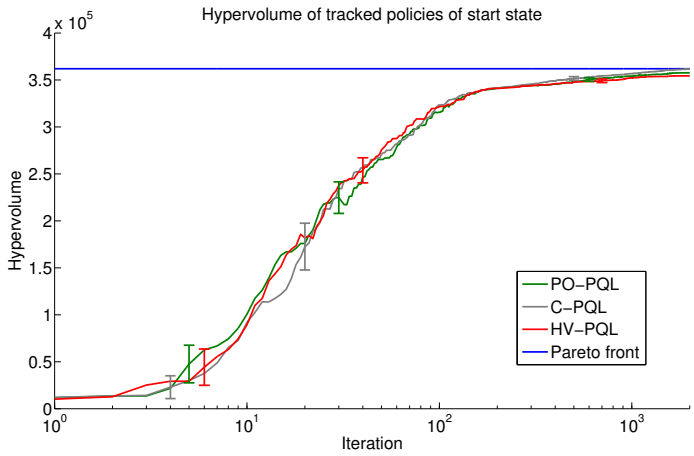


Figure 6.19: We see that the hypervolume of the tracked policies in the PBST environment is very similar to the hypervolume of the learned policies, which means that the learned policies are also retrievable.

The results on the train and test phases are depicted in Figures 6.18 and 6.19, respectively.[4] In Figure 6.19, we depict the hypervolume of the tracked policies of the start state by applying the greedy policies. As the environment has both deterministic reward and transition schemes, the performance of the different set evaluation mechanisms is almost identical. The tracking algorithm performs very well and the graph is almost identical to Figure 6.18 as in the previous environment.

6.3.7 Experimental comparison

In the previous section, we analysed the performance of Pareto Q-learning in combination with the three set evaluation mechanisms. In this section, we conduct an empirical comparison of the algorithms to several single-policy and multi-policy MORL algorithms. We conduct this experimental comparison on the Deep Sea Treasure environment.

The Deep Sea Treasure environment

The Deep Sea Treasure (DST) is a standard MORL benchmark instance that has been the subject of many performance evaluations in MORL. A general description of the environment can be found in Section 4.3.4. In Figure 6.20, we denote the hypervolume during the test phase of the algorithm with Pareto, cardinality and hypervolume set evaluations,

[4] In the PBST environment, the reference point for the hypervolume calculation in both *HV-PQL* and the performance assessment is specified to $(-25, 0, -120)$ after observing the reward scheme.

i.e., PO-PQL, C-PQL and HV-PQL, respectively. The reference point for the hypervolume calculation is specified to $(-25, 0)$.

Furthermore, we also evaluate a naive random strategy, a multi-policy algorithm and two single-policy algorithms. The multi-policy algorithm is the multi-objective Monte-Carlo tree search algorithm by Wang and Sebag (2013). Recall that this algorithm incrementally builds and explores a search tree of optimistic vectorial estimates. We implemented MO-MCTS-DOM, i.e., the version that relies on the Pareto dominance relation to scalarize the multi-objective value function, since this alternative is the best choice in terms of runtime and performance (Wang and Sebag, 2013). The two single-policy algorithms employ the linear and Chebyshev scalarisation functions of Chapter 4. These single-policy algorithms are evaluated using the configuration specified in Section 6.3.5. In the following paragraph, we highlight the performance of each algorithm individually.

Random The first algorithm we analyse is a random strategy. This algorithms serves as a baseline since it does not use any learning principles but naively applies a random policy. As expected, this algorithm learns slowly and reaches a hypervolume of 316. The 95% confidence interval of this algorithm is remarkably low since it always tends to find the same solutions, more precisely, the policies that are located close to the start state. The treasures that are far away are usually not discovered.

Linear scalarised MORL Since the environment contains 10 Pareto optimal solutions, the linear scalarised MORL algorithm is run with 10 uniformly distributed weight vectors to allow for a fair comparison to the multi-policy algorithms. More precisely, the continuous weight range of $[0, 1]$ is uniformly discretised with steps of $\frac{1}{10-1}$ while satisfying $\sum_{o=1}^{m} \mathbf{w}_o = 1$. Each of these weights is then used in an individual execution of the scalarisation algorithm and its results are collected in order to obtain a set of sampled policies in the test phase. Thus, every 10 iterations (episodes), we calculate the hypervolume of the 10 greedy policies of each of those iterations. We note that, although we have a uniform spread of weights, the algorithm only manages to retrieve a hypervolume of 768 which is in correspondence with our results from Chapter 5. When we take a closer look at the results obtained, we see that the algorithm learns fast but after 800 iterations, only the optimal policies with return $(-1, 1)$ and $(-19, 124)$ for the time and treasure objectives, respectively, are sampled. This is shown by the 95% confidence intervals that become zero. This is to be expected as the Pareto front is non-convex and, hence, a linear combination of the objectives then can only differentiate between the extreme policies of the Pareto front, as highlighted in Section 4.2. Therefore, the linear scalarised MORL algorithm converges to either of the optimal policies with return $(-1, 1)$ or $(-19, 124)$.

Chebyshev scalarised MORL This non-linear scalarisation function is equipped with the same set of weight vectors as the linear scalarisation function. As we see in the graph, the Chebyshev scalarised MORL algorithm learns slower than the linear scalarised MORL algorithm but is able to surpass its performance after around 1700 iterations. However,

as we have seen in Section 4.3, the Chebyshev scalarisation function is not guaranteed to converge to a Pareto optimal policy in a value-iteration approach. Nevertheless, we see that the algorithm performs acceptably in practice as it learns almost at the same speed as the linear scalarised MORL algorithm and attains a bigger hypervolume of 860.36. This is a confirmation that this scalarisation function, despite its lack of theoretical guarantees, can be successful in MORL when employed as a heuristic. Possibly, a higher hypervolume value could be reached, if we would equip the algorithm with more intelligent weight parameters. In this regard, the mechanisms of Chapter 5 could act as a starting point.

MO-MCTS-DOM The multi-policy MO-MCTS algorithm that utilises the Pareto dominance relation learns fast in the first 500 iterations but is, generally speaking, slower compared to the single-policy strategies. Earlier results of the algorithm indicate that the search tree algorithms requires substantial amounts of training time before a large set of policies is retrieved. This is because the algorithm builds an archive of non-dominated tree walks and, in the best case, appends only one solution at the end of every iteration. Therefore, the algorithm intrinsically only learns multiple policies in an implicit manner, while wasting information on explicit trade-offs as the interaction with the environment progresses. In the experiments in Wang and Sebag (2013), around 15600 iterations are needed for MO-MCTS-DOM to converge to the set of optimal solutions, which is significantly large for this environment. Therefore, considering 3000 iterations, allowed us only to slowly reach a final performance comparable to the Chebyshev scalarisation function.

Pareto Q-learning So far, no algorithms managed to sample the entire Pareto front. This was a result of either the shape of the Pareto front, the assignment of the weight vectors or an inefficient use of the available samples. Pareto Q-learning is not biased by the first two aspects, but treats every Pareto optimal solution equally in the bootstrapping process. Additionally, it explicitly learns multiple policies in a simultaneous fashion to allow for an efficient use of the available samples. Taking these principles into account, we see that in early learning phases, i.e, iteration 0 to 450, regardless of the set evaluation mechanisms, Pareto Q-learning learns notably faster than the other MORL algorithms. This is because PQL can simultaneously combine and store information about multiple optimal policies while the single-policy learners look only for an individual policy that matches its a priori preferences, thereby discarding considerable amounts of information. In the end, regardless of the set evaluation mechanisms used, Pareto Q-learning surpasses the performance of the other MORL algorithms. In this experiment, the Pareto set evaluation mechanism performs a bit better than the other set evaluation mechanisms and samples every element of the Pareto front.

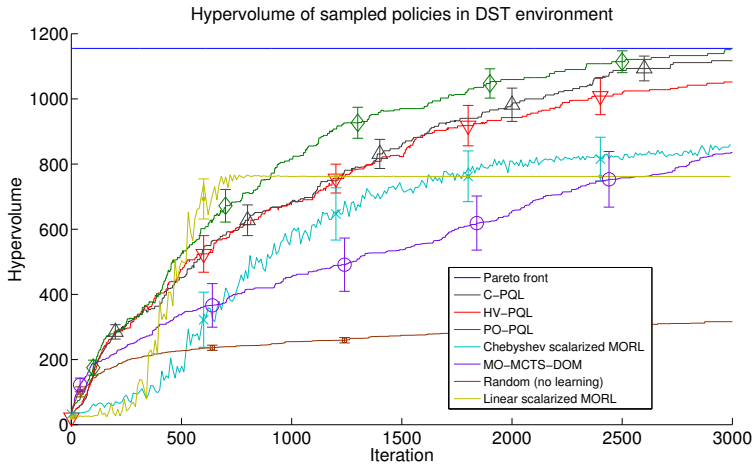


Figure 6.20: The results on the Deep Sea Treasure environment. We compare a random strategy, a multi-policy MORL algorithm named MO-MCTS-DOM and two single-policy MORL algorithms that use a linear and Chebyshev scalarisation function. These algorithms are then evaluated to Pareto Q-learning with the three set evaluation mechanisms. We note that, regardless of the set evaluation mechanisms, PQL obtained better performance than the multi-policy and single-policy MORL algorithms and in the end PO-PQL sampled the entire Pareto front. For a more in-depth analysis of the results, we refer to Section 6.3.7.

6.3.8 Conclusions

The main contribution of this section is the novel Pareto Q-learner algorithm that learns deterministic non-stationary non-dominated multi-objective policies for episodic environments with a deterministic as well as stochastic transition function. To the best of our knowledge, PQL is the first multi-policy temporal difference algorithm that allows to learn the entire Pareto front, and not just a subset. The core principle of our work consists of keeping track of the immediate reward vectors and the future discounted Pareto dominating vectors separately. This mechanism provides a neat and clean solution to update sets of vectors over time.

In a reinforcement learning algorithm, the exploration and exploitation trade-off is crucial. Therefore, we developed three evaluation mechanisms that use the $\hat{Q}_{set}$'s as a basis for action selection purposes during learning. We name them set evaluation mechanisms. The current set evaluation mechanisms rely on basic multi-objective indicators to translate the quality of a set of vectors into a scalar value. Based on these indications, local action selection is possible during the learning phase. Currently, we have combined PQL with a cardinality, hypervolume and Pareto indicator. We have seen that the Pareto indicator performed the best on average as it treats every Pareto optimal solution equally.

The cardinality and hypervolume set evaluation measures rate the actions also on additional criteria than the Pareto relation to provide a total order. We have seen that in more complex environments, these set evaluation mechanisms bias the action selection too much in order to learn the entire Pareto front. Nevertheless, it could be that the user is not interested in sampling the entire Pareto front but is looking for particular policies that satisfy certain criteria. For instance, other quality indicators such as the spread indicator (Van Veldhuizen and Lamont, 1998) could be used to sample policies that are both Pareto optimal and well-spread in the objective space. This can straightforwardly be incorporated in our framework.

Additionally, we also conducted empirical evaluations on a benchmark instance and we compared Pareto Q-learning's performance to several single-policy MORL algorithms. We have seen that selecting actions that are locally dominating does not guarantee that the overall combination of selected actions in each state, i.e., the policy, is globally Pareto optimal. As a solution, we proposed a mechanism that *tracks* a given return vector, i.e., we can follow a selected expected return consistently from the start state to a terminal state in order to collect the predicted rewards.

To summarise, we note that PQL (1) can learn the entire Pareto front under the assumption that each state-action pair is sufficiently sampled, (2) while not being biased by the shape of the Pareto front or a specific weight vector. Furthermore, we have seen that (3) the set evaluation mechanisms provide indicative measures to explore the objective space based on local action selections and (4) the learned policies can be tracked throughout the state and action space.

6.4 Summary

In this chapter, we investigated multi-objective reinforcement learning algorithms for simultaneously learning multiple policies. In this light, we have proposed two algorithms for both single state as well as multi state environments. The first algorithm is the multi-objective hierarchical optimistic optimisation algorithm that constructs a binary tree over a continuous actions space. The second algorithm extends standard Q-learning for discovering multiple trade-off policies at once for sequential decision making problems. Both algorithms share commonalities in the fact that they operate on sets of Pareto non-dominated vectorial estimates.

7 | Applications of multi-objective reinforcement learning

The contributions of the previous chapters concentrated on theoretical foundations of single-policy and multi-policy MORL. we will now investigate how we can apply these algorithms on a simulation environment, representing a real-life application. This environment simulates the filling phase of a wet clutch which is used in shifting and transmission systems of automotive vehicles.

In this chapter, we will analyse the application of the multi-objective hierarchical optimistic optimisation (MO-HOO) algorithm on the simulation environment in order to approximate its Pareto front. Before we proceed to the details of this practical exercise, it is necessary to properly introduce the details of this application domain in Section 7.1.

In more detail, we will present the following contributions this chapter:

- Analyse the simulation environment of the filling phase of a wet clutch
- Evaluate the performance of MO-HOO for multi-policy learning in the environment
- Outline directions for further research

This research has been published in Van Moffaert et al. (2014b).

7.1 The Filling Phase of a Wet Clutch

The problem considered in this paper is the efficient engagement of a wet clutch, i.e., a clutch where the friction plates are immersed in oil in order to smooth the transmission and increase the lifetime of the plates. Wet clutches are typically used in heavy duty

transmission systems, such as those found in off-road vehicles and tractors. In Figure 7.1, we depict a schematic representation of its functioning. Two sets of friction plates are connected to the input and the output shaft, respectively, such that, when they are pressed together, the torque generated by the motor, connected to the input shaft via a set of gears, is transmitted to the output shaft, which is connected to the wheels of the vehicle. The two sets of plates are free to translate axially, and a return spring keeps the clutch open: to close the clutch, the plates are pressed together by a hydraulic piston, which can be pushed against the plates by increasing the pressure of the oil in the clutch chamber. This pressure can in turn be controlled by the current input of an electromechanical servo-valve: by opening the valve, the clutch and supply line are filled up with oil, and the pressure increases until it is high enough to overcome the return spring force. As a result, the piston starts to move towards the plates. During this first part of the engagement, called the *filling phase*, no torque is transferred. The transmission only commences as the piston makes contact with the plates. The clutch then enters the *slip phase*, as the slip, defined as the difference in rotational speeds between the shafts, decreases. When the pressure is high enough, the output shaft is accelerated until it rotates synchronously with the input, and the slip reaches zero.

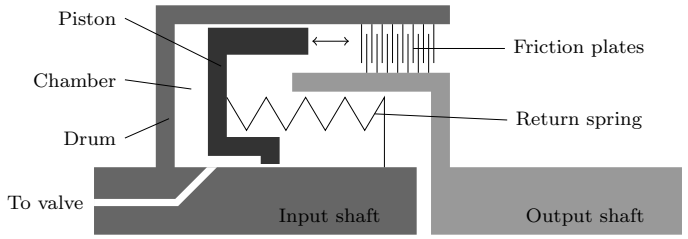


Figure 7.1: Schematic representation of a wet clutch (from Depraetere et al. (2010)).

However, it has been observed that the torque loss depends on the speed of the piston when this reaches the plates: the lower the speed, the lower the torque loss, and the smoother the engagement. In other words, there is a trade-off among the two objectives: on one hand, a very slow piston movement will result in a smooth engagement, which takes a long time; on the other hand, a fast piston will engage in a short time, but it will also cause a jerky engagement, with a large torque dip, and possibly damage the setup. These two objectives are conflicting and make a good testbed for evaluating multi-objective solution techniques.

The main issue with this setup is that there is no sensor providing information about the actual position of the piston, which can only be detected when the piston actually touches the plates. Given the available signals (oil pressure and temperature), there is no reference trajectory that can be tracked using classical control methods. In this sense, this setup is a good example of a limited feedback system. For this reason, wet clutches are commonly controlled in open loop, applying a signal which is either defined ad-hoc, or learned iteratively (Pinte et al., 2011).

A parametric form of such a signal (Zhong et al., 2011) can be found in Figure 7.2. First, a large current pulse is sent to the valve, in order to generate a high pressure level in the oil chamber of the clutch, which will allow the piston to overcome the resistance of the preloaded return spring, and accelerate towards the friction plates. After this pulse, a lower constant current is sent out, in order to decelerate the piston as it approaches the friction plates: before reaching this low value, the current decreases shortly to an even lower value, creating a small dent in the signal, which should act as a 'brake', limiting the speed of the piston. Finally, the current signal grows following a ramp with a low slope, closing the clutch smoothly. The signal is parametrised as follows: the first parameter (a) controls the duration of the initial peak, whose amplitude is fixed at the maximum level, while the last (d) controls the low current level, just before the engagement begins. The remaining parameters (b, c) control the shape of the 'braking' dent, while the final slope during the engagement phase is fixed. In our setting the signal consists of two parameters, i.e., only a and c are free, while $b = 0$, and $d = c$.

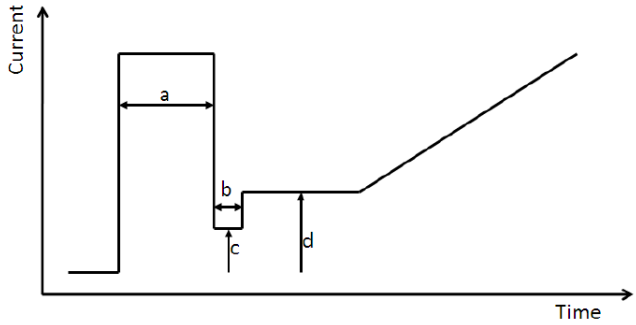


Figure 7.2: Parametric input signal from Zhong et al. (2011), with four parameters (a, b, c, d). All parameter ranges are mapped to the unit interval $[0, 1]$. In this setting, only two parameters can be controlled.

There currently is no reliable model of the whole engagement. However, an approximate but deterministic model is available for the filling phase, until the piston touches the plates. [1] Nonetheless, this model does not comprise the slip phase, which would allow to simulate and optimise the resulting torque loss. But, torque loss has been observed to depend on the speed of the piston upon engagement, and we can therefore minimise piston velocity in simulation.

[1] Developed in Simulink® by Julian Stoev, Gregory Pinte (FMTC), Bert Stallaert and Bruno Depraetere (PMA, Katholieke Universiteit Leuven).

7.2 Multi-policy analysis with MO-HOO

In this section, we analyse the MO-HOO algorithm of Section 6.2 on the wet clutch environment. Since the model of the wet clutch consists of a single state with a continuous but finite dimensional action space, the MO-HOO algorithm is a perfect fit for the job.

Before we continue to the actual results, it is necessary to have an intuition of the size and the shape of the Pareto front of this environment. In contrast to the relatively simple Schaffer 1 and Fonseca and Flemming functions of Section 6.2.3, where the identification of the continuous Pareto front is relatively easy, this is not the case for the wet clutch environment. In Figure 7.3, we denote the policies that are discovered through an exhaustive parameter sweep. First, we normalise both reward signals and transform them for maximisation purposes:

$$r_v = v, \tag{7.1}$$

$$r_t = (3 - t)/3. \tag{7.2}$$

We note that many of the parameters lead to dominated policies located in clustered suboptimal areas of the objective space. These dominated policies are denoted by a red colour. Additionally, we also see that some regions in the objective space are more dense than others. This is an indication of the level of difficulty of the problem since small changes in the decision space, i.e., the parameter values, imply big changes in the objective space. The exhaustive search only discovered 8 Pareto non-dominated policies, denoted by black squares, which are located in the top-right corner of the objective space. Since the Pareto front is continuous, there should exist other non-dominated policies connecting the black dots. The question is how to find them without wasting extensive resources on the dominated areas of the objective space.

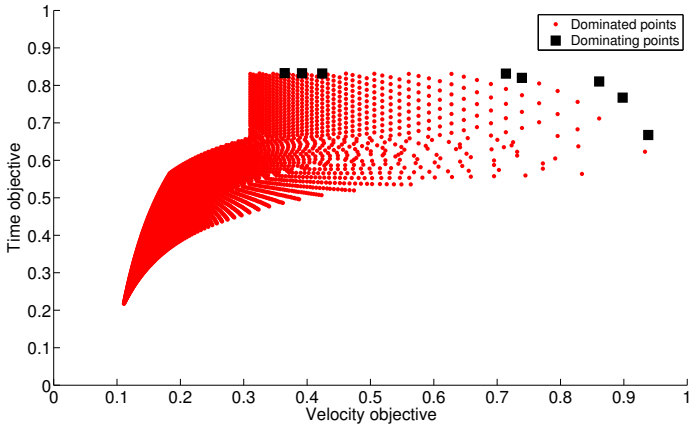


Figure 7.3: An exhaustive search of the parameter space of the wet clutch simulation environment discovered only 8 non-dominated policies.

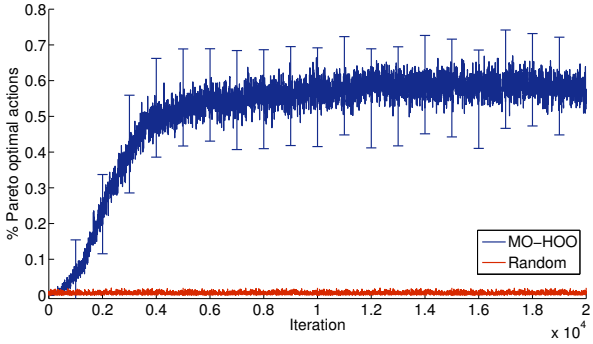


Figure 7.4: The sampling performance MO-HOO is compared to a naive random strategy. We note that the MO-HOO strategy approaches a performance of around 60%, although the standard deviation is quite large.

In Figure 7.4, we depict the results of the MO-HOO algorithm and a random algorithm on this simulation environment. More precisely, we analyse the distance over time between between the sampled policies and the 8 Pareto non-dominated policies retrieved by the exhaustive search method. We inspect the percentage of Pareto optimal actions at a distance of δ equal to 0.2 in the normalised objective space.

As expected on this complex environment, we see that randomly sampling the action space almost never leads to a Pareto optimal policy. We even note that at some iterations, the standard deviation is zero, meaning that in none of the 50 runs an element close to the Pareto front is sampled. The MO-HOO strategy also struggles on this environment and suffers from quite some variation. Eventually, it approaches a performance of 60%. Logically, the question arises why this performance does not reach 100% as time progresses. To answer that question, we first investigate the sampled policies in the objective space over time in Figure 7.5.

In Figure 7.5, the policies are colour-coded, with blue indicating policies obtained in early iteration cycles, and red representing policies sampled near the end of the process. In this plot, we notice several observations. For starters, we see that there appears a gradient from early, dominated policies to policies closer to the 8 non-dominated points found by the exhaustive search process. These points are denoted by black dots. We also note that MO-HOO obtains many optimal trade-off policies that lie in between the 8 initial elements of the Pareto front.

In Figure 7.6, we depict the sampled points in the action space and the corresponding tree depth. This visualisation confirms our intuition: on this complex task, the MO-HOO strategy is excessively exploring although a clear trend in the sampled points appears after already 4000 iterations. The tree also refines nodes at two particular parts of the action space and two peaks emerge since we are optimising two actions simultaneously.

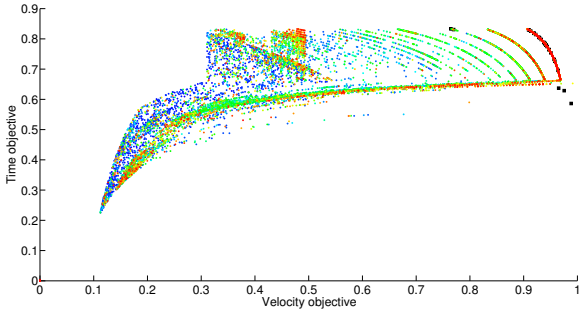


Figure 7.5: The sampled policies in the objective space during each of the 20000 iterations. The policies are colour-coded, with blue indicating policies obtained in early iteration cycles, and red representing policies sampled near the end of the process. In general, red policies are close to the Pareto front that was previously extracted, although it is not always the case.

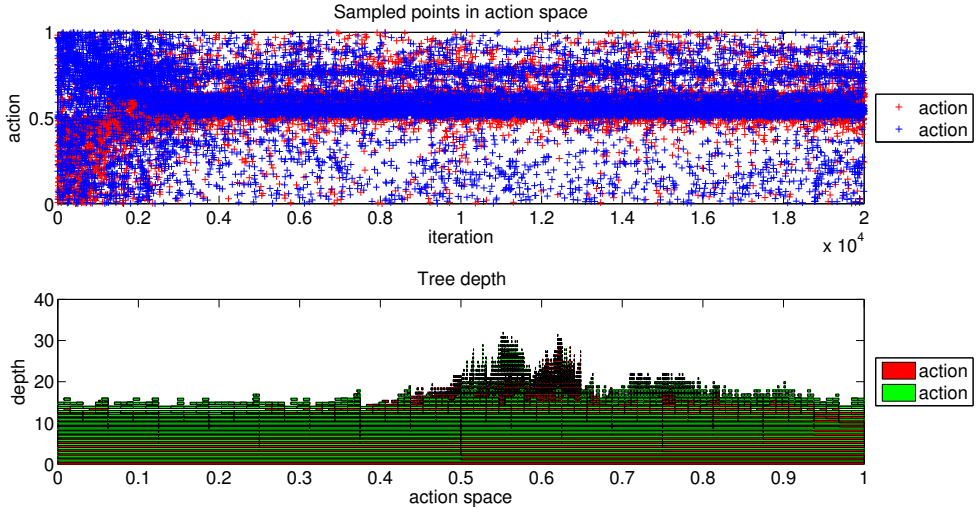


Figure 7.6: Both figures more closely analyse the performance of MO-HOO on the wet clutch environment. In the first subfigure, we depict the sampled points in the actions space. We see that for each of the signals a and c a trend in the action space is emerging over time, although there remains a substantial amount of variation. In the second subfigure, we see the evolution of the tree structure of the MO-HOO algorithm. Also in this figure, we note that the algorithm is exploring the entire action space for quite some time before it is confident enough to exploit a particular area.

The cause for this behaviour is not very clear and might have multiple sources. First, the wet clutch environment is a combination of multiple complex interacting systems which impedes the analysis of the underlying optimisation function. In contrast to the standard benchmark settings, such as the Schaffer 1 function, the wet clutch function is hard to grasp and to visualise. Therefore, it is possible that the function does not satisfy the assumption of the HOO strategy, which is that it should be locally Lipschitz differentiable. Another possibility is that the MO-HOO strategy itself is exploring too much since its estimates are too optimistic. This could be resolved by carefully tuning the ρ and ν_1 parameter values which define the level of optimism added to each estimate. A possibility would be to employ the F-Race algorithm (Birattari et al., 2010) to find the best values for these parameters that steer the quantity of exploration. Finally, it could also be an issue with the bounds the algorithm itself places on the estimates. Since there is no mechanism, i.e, no $\min$ operator, to bound the estimates and as a result the algorithm keeps exploring parameters that are suboptimal. A thorough analysis of the underlying function of the wet clutch could possibly identify the issue.

To conclude, we still note that the MO-HOO strategy discovers a wide set of non-dominated trade-off solutions since it obtained on average 357.21 Pareto non-dominated solutions at each run of the algorithm. As expected, the random strategy obtained only a small set of 7.66 non-dominated solutions out of 20000 samples, as depicted in Table 7.1.

Strategy	Cardinality
Random	7.66
MO-HOO	**357.21**

Table 7.1: Cardinality indicator for random exploration and MO-HOO.

7.3 Conclusions and future outlook

In this chapter, we analysed to which extent the theoretical algorithms behave on a complex and real-world application. This application comprises the filling phase of a wet clutch, which is a crucial part in a wide range of automotive vehicles.

We applied the MO-HOO strategy of Section 6.2 on a simulation environment, representing this setup. We have seen that the algorithm was struggling to find a good trade-off between exploring uncharted territory and exploiting its knowledge near the end the of the optimisation process. As a result, the results had a larger variation than on the benchmark environments but the algorithm still performed acceptably. Eventually, MO-HOO discovered no less than 357 non-dominated elements on average in each run, offering a wide and diverse set of compromise solutions to the decision maker. We believe that a closer investigation of the internals of the wet clutch environment could be beneficial to find better values for the configuration parameters of the MO-HOO strategy itself, although this remains an unanswered question for now.

8 | Conclusions

Reinforcement learning is the study of intelligent systems that learn to take optimal decisions in unknown and dynamic problem environments. Traditionally, this study aims to learn the optimal actions so as to maximise a single, numerical reward signal. In this light, we have illustrated some standard approaches such as the Q-learning algorithm in Chapter 2. As we have seen, reinforcement learning provides a rigorous theoretical framework for learning systems. However, this framework is based on the assumption that every action can be evaluated by a single, scalar feedback signal. As we have seen in Chapter 3, many real-world problems involve the simultaneous optimisation of multiple, possibly conflicting, objectives. An example is the optimisation of the return and its risk in portfolio optimisation for stock markets (Hassan, 2010). It is clear that a risky policy has the potential of a high return of investment while at the same time a high probability in money losses as well. Therefore, there is no single optimal solution but a set of equally optimal compromise solutions that balance these objectives. This set is referred to as the Pareto front. In this case, the problem the agent tries to solve is considered a multi-objective reinforcement learning problem.

In this dissertation, we study the development and the evaluation of MORL algorithms that extend the standard, single-objective framework to discover a number of trade-off solutions. In Section 8.1, we highlight our main contribution, presented throughout this thesis. Subsequently, in Section 8.2, we formalise additional remarks and limitations of our newly developed algorithms which on their turn serve as a starting point for outlooks for future research in Section 8.3.

8.1 Contributions

In this section, we summarise the main contributions of this dissertation. To thoroughly understand the research domain, we presented a necessary overview of the foundations of

standard, single-objective reinforcement learning in Chapter 2. This background is needed to support the reader to fully comprehend the domain of multi-objective reinforcement learning in Chapter 3. In that chapter, we discussed current state-of-the-art strategies for multi-objective reinforcement learning and we proposed a taxonomy that classifies these strategies on two main aspects. The first aspect considers whether the algorithm retrieves a single solution or a set of solutions. As a result, the algorithm is either a single-policy or a multi-policy approach. The second aspect is based on the type of scalarisation function used in the process, i.e., is the function linear or merely monotonically increasing. In the following paragraphs, we highlight our principal contributions that surpass the state-of-the-art in the field of MORL.

In Chapter 4 we considered MORL algorithms that learn only a single trade-off solution at a time. These algorithms use scalarisation functions to reduce the dimensionality of the multi-objective feedback signal to a single, scalar measure. To accommodate for these functions, we have presented a general framework for scalarised MORL algorithms. In this framework, the traditional Q-values are extended to $\mathbf{Q}$-vectors that store an estimate for every objective separately. In the action selection phase, the scalarisation function is applied on the $\mathbf{Q}$-vectors to acquire a single, scalarised estimate for a state-action pair. This scalarised estimate can on its turn be used for action selection purposes, such as for instance ϵ-greedy. Although the idea is relatively simple, the choice of the scalarisation function is crucial. We have elaborated on the standard linear scalarisation function and investigated why it is guaranteed to converge to a Pareto optimal solution and why it cannot retrieve every Pareto optimal solution. Additionally, we have researched to what degree the monotonically increasing Chebyshev scalarisation function can also be applied in MORL. In both a theoretical and empirical analysis we have seen that the Chebyshev function lacks additivity, which is a crucial requirement for reinforcement learning algorithms that rely on the Bellman equation. However, the Chebyshev scalarisation function can still be used as a heuristic without possessing any convergence guarantees. We compared both scalarisation functions on three commonly accepted benchmark environments on several performance measures.

Subsequently, in Chapter 5, we analysed how the weight parameters of the scalarisation functions relate to specific parts of the objective space. We have seen that the mapping from weight space to objective space is non-isomorphic, meaning that it is far from trivial to define an appropriate weight vector that retrieves a desired compromise solution. In this regard, we have proposed two classes of algorithms that adaptively explore both discrete and continuous Pareto fronts. The algorithm that can be used in a discrete setting constructs a tree over the weight space. Each iteration the tree is iteratively refined to cover finer parts of the weight space until all solutions are found. This algorithm is evaluated on two testing environment with two to three objectives. In the case the Pareto front is continuous and, in essence, infinite, the goal is to find as fast as possible a diverse set of optimal solutions that provide a careful coverage over the Pareto front. This adaptive weight algorithm (AWA) can be combined with several distance metrics, such as the hypervolume measure and a gap distance. This mechanism, together with all

its variants, is tested on a smart camera environment where the goal is to maximise the tracking confidence of moving objects while minimising the computational overhead.

In Chapter 6 we presented the pinnacle of our contributions. In this chapter, we no longer use scalarisation functions to reduce the dimensionality, and hence also the difficulty of the problem. Instead, we extend the core principles of reinforcement learning framework to simultaneously learn a set of trade-off solutions. The first algorithm we proposed in this context is the multi-objective hierarchical optimistic optimisation (MO-HOO) algorithm which is tailored for single-state environments with a continuous action space, as is the case in a lot of engineering applications. The MO-HOO algorithm builds a tree of optimistic estimates over the action space and propagates a set of Pareto non-dominated estimates to the root node. Based on these estimates, the algorithm continues the search to promising parts of the action space that result in optimal and diverse solutions in the objective space. This algorithm is tested on two optimisation functions, commonly used in evolutionary multi-objective optimisation. For multi-state environments that require a bootstrapping procedure, we proposed the Pareto Q-learning algorithm. Pareto Q-learning is the first temporal difference-based multi-policy algorithm that does not employ a linear scalarisation function and can therefore learn the entire Pareto front and not just a subset. The main novelty of the Pareto Q-learning algorithm is the set-based bootstrapping rule that separates the average immediate reward and the set of Pareto non-dominated vectorial estimates. Pareto Q-learning can be combined with several quality measures to accommodate for action selection strategies, such as the hypervolume and the cardinality quality indicators.

In Chapter 7, we moved beyond the academic benchmark environments and tested multi-policy MORL algorithms on a simulation environment of a real-world problem. This problem concerns the filling phase of a wet clutch, a transmission unit that can be found in many automotive vehicles. In this setup, the goal of the agent is to find an engagement that is both fast and smooth, which are in essence conflicting objectives. We evaluated the MO-HOO algorithm to learn a set of trade-off solutions at once. We have seen the task introduces an amplified level of complexity but the algorithm still managed to obtain on average 357 trade-off solutions in a single run.

8.2 Remarks

Throughout this thesis, we have presented several algorithms for learning in multi-criteria environments. Many of these algorithms are novel in their specific field and required changes to the theoretical reinforcement learning framework on a detailed conceptual level. For instance, the MO-HOO algorithm of Chapter 6 is, to the best of our knowledge, the first multi-objective $\mathcal{X}$-armed bandit solver. On the one hand side, this algorithm extends the boundaries of the MORL field but on the other hand side, the novelty of the algorithm complicates a rigorous experimental comparison to other, specific algorithms for the same type of problems. One possibility is to improve the experimental evaluation of the algorithm is to compare it to evolutionary multi-objective algorithms. However, for the comparison to

be fair, one needs to take into account some aspects. For instance, evolutionary algorithms inherently evolve a population of solutions. Therefore, they mimic a multi-agent principle where agents can cooperate to improve the overall quality. Additionally, the size of the population plays a crucial role in the process and influences the performance as well.

A second remark is related to the convergence guarantees of the MO-HOO algorithm. In that algorithm, we adjusted the internal mechanisms of the HOO tree to propagate Pareto non-dominated estimates to the root node. Since our estimates are not averaged over subtrees and there is no alternative multi-objective $\min$ operator, we cannot assume MO-HOO to have the same convergence rate and upper bounds for the regret as the HOO strategy. Although the algorithm seems to perform well on the three experiments we carried out in this dissertation, a thorough theoretical analysis of the mechanism is still needed.

Thirdly, due to time constraints, we did not perform a rigorous

8.3 Outlooks for future research

Finally, we conclude the thesis by listing outlooks for future research that can open the door to new advances in the multi-objective reinforcement learning domain:

- *Alternative distance metrics for AWA algorithms*: In Chapter 5, we have proposed several distance measures that can be used to guide the search to optimal and well-spread solutions. Although, the list of possible distance measures is quite elaborate, there is no single best metric that will be the ideal solution approach for every application domain. A possible next step would be to directly incorporate the genuine multi-objective quality indicators such as the spread indicator (Van Veldhuizen and Lamont, 1998) directly in the internal workings of the adaptive weight algorithm.
- *Sampling the Pareto front with AWA algorithms*: One could extend the reach of the adaptive weight algorithms by extending the scalarisation principle. As this mechanism currently relies on a convex combinations of the objectives, only policies on the convex hull are retrievable. To also empower the discovery of policies in non-convex regions of the Pareto front, one could try to combine AWAs with the adaptive weighted sum principle in Kim and de Weck (2005) and Yliniemi and Tumer (2014), where the objective space of the original problem is transformed to a convex shape. In this transformed Pareto front, the linear scalarisation function can find solutions that correspond to solutions in non-convex parts of the original objective space.
- *Tighter bounds on MO-HOO*: the MO-HOO algorithm of Chapter 6 is based on the HOO algorithm which uses a combination of traditional $\min$ and $\max$ operator to place bounds on the level of optimism of the estimates. In MO-HOO, the $\max$ operator is replaced by a union operator that filters out the Pareto non-dominated estimates. No multi-objective variant is found for the $\min$ operator and as a result, the estimates might become too optimistic, resulting in an algorithm that explores too much. In future work, the specification of a multi-objective $\min$ operator could place a stricter bound on the $\mathcal{B}$-vectors in order to make them less optimistic.

However, it is not clear to determine what would be the logical outcome of applying a multi-objective $\min$ operator on a $\boldsymbol{\mathcal{U}}$-vector and a set of non-dominated $\boldsymbol{\mathcal{B}}$-vectors. Since there is no total order, defining a $\min$ operator that can deal with multiple Pareto incomparable vectors remains an open research question.

- *Alternative action evaluation indicators for Pareto Q-learning*: In the Pareto Q-learning algorithm of Chapter 6, three different action evaluation mechanisms are proposed. These are the hypervolume, cardinality and the Pareto set evaluation mechanisms which quantify the quality of a set of $\mathbf{Q}$-vectors. These mechanisms were designed to treat every Pareto optimal solution equally (to a certain extent). However, it could be that the decision maker is not interested in sampling the entire Pareto front but that he is looking for particular policies that satisfy certain criteria. For instance, other quality indicators such as the spread indicator (Van Veldhuizen and Lamont, 1998) could be used to sample policies that are both Pareto optimal and well-spread in the objective space. In future research, alternative set evaluation mechanisms could be investigated to steer the search process.
- *Pareto Q-learning for infinite-horizon problems*: In Pareto Q-learning, the Q_{set}'s grow according to the size of the Pareto front. Therefore, Pareto Q-learning is primarily designed for episodic environments with a finite number of Pareto optimal policies. To make the algorithm practically applicable for infinite horizon problems with a large value for the discount factor, we have to consider that all states can be revisited during the execution of an optimal policy. Upon revisiting a state, a different action that is optimal w.r.t. other criteria can be chosen. As explained in Mannor and Shimkin (2001), this offers a possibility to steer the average reward vector towards a target set using *approaching* policies. Alternatively, we could reduce the number of learned policies by using a consistency operator to select the same action during each revisit of some state, similar to the work of Wiering and de Jong (2007) for multi-criteria dynamic programming.
- *Pareto Q-learning on continuous states*: Currently, Pareto Q-learning is limited to a tabular representation where each state-action pair stores a Q_{set}. In order to make Pareto Q-learning applicable to real-life problems or ergodic environments, these sets should also be represented through function approximation. A possible idea is to fit the elements in each set through a geometric approach, such as for instance ordinal least-squares. If the environment would consist of two or three objectives, we would be fitting the vectors on a curve or a plane, respectively. In that case, we would be learning the shape of the Pareto front through local interactions that each update parts of this geometric shape. Therefore, this research direction could pave the way for multi-policy MORL in real-world problem environments.
- *Risk-sensitive learning with Pareto Q-learning*: Although Pareto Q-learning is primarily designed for multi-objective problems. It could also be employed to augment standard single-objective problems. Through a principle called *multi-objectivisation*, additional, artificial objectives can be constructed to improve the performance on the original objective, as measured by solution quality, time to solution, or some other measure (Knowles et al., 2001). In this way, one could solve a practical problem of

reinforcement learning in real-world applications, i.e., in those cases, algorithms that solely focus on maximising the mean return might be inappropriate as they do not account for the variability of their solutions. Thereby, a variability measure could be included to accommodate for a risk-sensitive setting, where the decision maker can explicitly define the tolerated level of deviation of the observed return from the expected return. In Van Moffaert et al. (2015), we scratched the surface of this idea by learning a set of optimal trade-off policies that balance mean reward and its variance.

- *Many-objective optimisation*: The algorithms presented throughout this dissertation were developed for multi-objective problems. In practice this means the environments deal with two to three objectives. Many-objective optimisation is another research topic that moves beyond this point and deals with the optimisation of four or more objectives (Deb and Jain, 2014; Jain and Deb, 2014). This branch poses great challenges to the classical approaches since the number of Pareto incomparable solutions grows when the number of objectives increases as well (Winkler, 1985). Nonetheless, the underlying mechanisms of our algorithms, such as the hypervolume and the Pareto dominance relation, are without adaptation also theoretically applicable in these many-objective environments. However, a huge amount of Pareto incomparable solutions will unquestionably affect the running time of the algorithms in practice since the space of possible solutions can become very large. In future work, researchers could investigate to what degree these algorithms scale up to many-objective problems. A possible idea to shrink down the huge solution space would be to employ user-defined preference functions that emphasise certain parts of the objective space.

8.4 Overview

In this dissertation, we have presented several multi-objective reinforcement learning algorithms for many different situations. For the sake of clarity, in Figure 8.1, we summarise our algorithmic contributions based on three aspects. These aspects are (1) whether the decision maker wants to learn a single or multiple policies, (2) whether or not convex solutions suffice (or are also non-convex solutions of interest?) and (3) whether the problem consists of a continuous or a discrete action range.

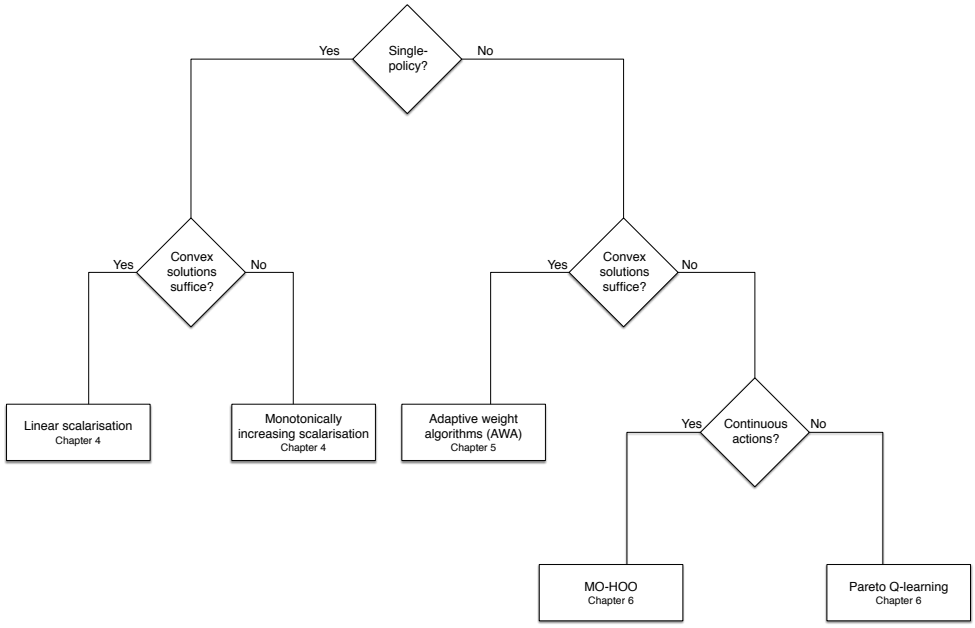


Figure 8.1: An overview of the different algorithmic contributions in this dissertation, depending on the type of problem and whether convex solutions suffice for the decision maker.

List of Selected Publications

International Journals and Lecture Notes

1. K. Van Moffaert, A. Nowé. *Multi-Objective Reinforcement Learning using Sets of Pareto Dominating Policies.* Journal Of Machine Learning Research, vol. 15, 2014, p 3483-3512.

2. K. Van Moffaert, M. M. Drugan, A. Nowé. *Hypervolume-based Multi-Objective Reinforcement Learning.* In Proceedings of the 7th International Conference on Evolutionary Multi-Objective Optimization (EMO), 2013, Lecture Notes in Computer Science 7811, 2013, p 3483-3512.

Proceedings of Conferences with International Referees

1. K. Van Moffaert, K. Van Vaerenbergh, P. Vrancx, A. Nowé. *Multi-Objective X-Armed Bandits.* In Proceedings of the IEEE World Congress on Computational Intelligence (WCCI), 2014, p 2331-2338.

2. K. Van Moffaert, T. Brys, A. Chandra, L. Esterle, P. Lewis, A. Nowé. *A Novel Adaptive Weight Selection Algorithm for Multi-Objective Multi-Agent Reinforcement Learning.* In Proceedings of the IEEE World Congress on Computational Intelligence (WCCI), 2014, p 2306-2314.

3. K. Van Moffaert, M. M. Drugan, A. Nowé. *Scalarized Multi-Objective Reinforcement Learning: Novel Design Techniques.* In Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2013, p 191-199.

4. K. Van Moffaert, Y. M. De Hauwere, P. Vrancx, A. Nowé. *Reinforcement Learning for Multi-Purpose Schedules.* In Proceedings of the 5th International Conference on Agents and Artificial Intelligence (ICAART), 2013, p 203-209.

Extended abstracts

1. T. Brys, K. Van Moffaert, M. M. Drugan, A. Nowé, M. Taylor. *Adaptive Objective Selection for Correlated Objectives in Multi-Objective Reinforcement Learning.* In Proceedings of the 13th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), p 1349-1350.

2. K. Van Moffaert, M. M. Drugan, A. Nowé. *Learning Sets of Pareto Optimal Policies.* In Proceedings of the AAMAS 2014 Workshop on Adaptive and Learning Agents (ALA), 2014.

3. K. Van Moffaert, M. M. Drugan, A. Nowé. *Multi-Objective Reinforcement Learning.* In Proceedings of the European Workshop on Reinforcement Learning (EWRL Dagstuhl Seminar 13321), 2013, p13-14.

List of abbreviations

AI	Artificial intelligence
AN-TPLS	Adaptive normal two-phase local search
AN-TPRL	Adaptive normal two-phase reinforcement learning
AN-TPLS-HV	Adaptive normal two-phase local search using the hypervolume distance
AN-TPRL-HV	Adaptive normal two-phase reinforcement learning using the hypervolume distance
AN-TPRL-OHV	Adaptive normal two-phase reinforcement learning using the overlapping hypervolume distance
AWA	Adaptive weight algorithm
BST	Bountiful Sea Treasure environment
CH	Convex hull
CHVI	Convex hull value iteration
CON-MODP	Consistent multi-objective dynamic programming
C-PQL	Pareto Q-learning using cardinality-based action selection
DST	Deep Sea Treasure environment
DP	Dynamic programming
EMO	Evolutionary multi-objective optimization
GD	Generational distance

HOO	Hierarchical optimistic optimization
HV	Hypervolume
HV-PQL	Pareto Q-learning using hypervolume-based action selection
IGD	Inverted generational distance
MCTS	Monte-Carlo tree search
MDP	Markov decision process
MOFQI	Multi-objective fitted Q-iteration
MOMDP	Multi-objective Markov decision process
MO-MCTS	Multi-objective Monte-Carlo tree search
MO-HOO	Multi-objective hierarchical optimistic optimization
MORL	Multi-objective reinforcement learning
ND	Non-dominated operator
PF	Pareto front
PGPE	Policy gradient with parameter exploration
PO-PQL	Pareto Q-learning using Pareto dominance-based action selection
PQL	Pareto Q-learning
RA-TPLS	Regular anytime two-phase local search
RA-TPRL	Regular anytime two-phase reinforcement learning
RA-TPRL-DIST	Regular anytime two-phase reinforcement learning using the Euclidean distance
RA-TPRL-HV	Regular anytime two-phase reinforcement learning using the hypervolume distance
RA-TPRL-OHV	Regular anytime two-phase reinforcement learning using the overlapping hypervolume distance
RL	Reinforcement Learning
TD	Temporal difference
TPLS	Two-phase local search
TPRL	Two-phase reinforcement learning
OHV	Overlapping hypervolume

List of symbols

Single-objective reinforcement learning	
$a(t)$	Selected action at timestep t
$A(s)$	Action set in state s
α	Learning rate
γ	Discount factor
ϵ	Exploration parameter of the ϵ-greedy action selection strategy
t	Timestep
$s(t)$	State at timestep t
$S = s^1, \ldots, s^N$	Set of states
$r(t+1)$	Observed immediate reward at timestep $t+1$
$T(s'\|s,a)$	Probability of arriving in state s' after selecting action a in state s
$R(s,a,s')$	Expected reward associated to transitioning from state s with action a to state s'
τ	Temperature parameter of the Softmax action selection
R_t	The return up to timestep t
π	Policy
$\pi(s,a)$	Probability of selecting an action a in state s
Π	Space of policies
$V^\pi(s)$	The value of state s under policy π
$V^*(s)$	The value of state s under an optimal policy
$Q^\pi(s,a)$	The value of action a in state s under policy π
$Q^*(s,a)$	The value of action a in state s under an optimal policy
$\hat{Q}(s,a)$	The estimated value of action a in state s

Multi-objective reinforcement learning	
$\mathcal{F}$	Multi-objective optimization function
m	Number of objectives
$\mathbf{X}$	The space of decision vectors
$\prec$	Strict Pareto dominance
$\preceq$	Weak Pareto dominance
$\parallel$	Pareto incomparable
$\not\prec$	Pareto non-dominated
$\mathbf{R}(s,a,s')$	Expected reward vector associated to transitioning from state s with action a to state s'
$\mathbf{V}^{\pi}(s)$	The vector value of state s under policy π
$\mathbf{V}^{*}(s)$	The vector value of state s under an optimal policy
$\mathbf{Q}^{\pi}(s,a)$	The vector value of action a in state s under policy π
$\mathbf{Q}^{*}s,a$	The vector value of action a in state s under an optimal policy
$\hat{\mathbf{Q}}(s,a)$	The estimated vector value of action a in state s
$SQ(s,a)$	The scalarized value of action a in state s
$CH(\Pi)$	The convex hull of a set of policies Π
$PF(\Pi)$	The Pareto front of a set of policies Π
$\mathbf{z}^{*}$	The attraction point of the Chebyshev scalarisation function
$\mathbf{w}$	The weight vector of a scalarisation function
$\mathcal{X}$	The continuous action space of an $\mathcal{X}$-armed bandit problem
(h,i)	A node of the tree of the HOO strategy at index i and depth h
$\mathcal{P}_{h,i}$	The area of the action space corresponding to node (h,i)
$\mathcal{U}_{h,i}(n)$	The U-value of node n located at index i and depth h
$\boldsymbol{\mathcal{U}}_{h,i}(n)$	The $\mathcal{U}$-vector of node n located at index i and depth h
$\mathcal{B}_{h,i}(n)$	The $\mathcal{B}$-value of node n located at index i and depth h
$\boldsymbol{\mathcal{B}}_{h,i}(n)$	The $\mathcal{B}$-vector of node n located at index i and depth h
$\hat{\mu}_{h,i}(n)$	The estimated average reward of node n located at index i and depth h
$\mathcal{T}_{h,i}(n)$	The number of times node n located at index i and depth h is sampled
$\hat{Q}_{set}(s,a)$	A set of $\mathbf{Q}(s,a)$ vectors
$\overline{\mathfrak{R}}(s,a)$	The average immediate reward of taking action a in state s
$ND_t(s,a)$	The set of Pareto non-dominated vectors of the state reached after selecting action a in state s
$n(s,a)$	The number of times action a is selected in state s
$F_{s,a}^{s'}$	The observed frequency of reaching state s' after selecting a in state s

Bibliography

Z Aimin, J. Yaochu, Z. Qingfu, B. Sendhoff, and E. Tsang. Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 892–899, 2006. doi: 10.1109/CEC.2006.1688406.

M. J. Alves and J.N. Climaco. An interactive method for 0-1 multiobjective problems using simulated annealing and tabu search. *Journal of Heuristics*, 6(3):385–403, 2000.

M. Aruldoss, T.M. Lakshmi, and V.P. Venkatesan. A survey on multi criteria decision making methods and its applications. *American Journal of Information Systems*, 1(1): 31–43, 2013. doi: 10.12691/ajis-1-1-5. URL http://pubs.sciepub.com/ajis/1/1/5.

J.-Y. Audibert, R. Munos, and C. Szepesvári. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.*, 410(19):1876–1902, April 2009. ISSN 0304-3975. doi: 10.1016/j.tcs.2009.01.016. URL http://dx.doi.org/10.1016/j.tcs.2009.01.016.

P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Mach. Learn.*, 47(2-3):235–256, May 2002. ISSN 0885-6125. doi: 10.1023/A:1013689704352. URL http://dx.doi.org/10.1023/A:1013689704352.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.*, 32(1):48–77, January 2003. ISSN 0097-5397. doi: 10.1137/S0097539701398375. URL http://dx.doi.org/10.1137/S0097539701398375.

B. Awerbuch and R. Kleinberg. Online linear optimization and adaptive routing. *J. Comput. Syst. Sci.*, 74(1):97–114, February 2008. ISSN 0022-0000. doi: 10.1016/j.jcss.2007.04.016. URL http://dx.doi.org/10.1016/j.jcss.2007.04.016.

T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996. ISBN 0-19-509971-0.

T. Bäck. *Evolutionary algorithms in theory and practice - evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996. ISBN 978-0-19-509971-3.

L. Barrett and S. Narayanan. Learning all optimal policies with multiple criteria. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 41–47, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390162.

L.S. Batista, F. Campelo, F.G.. Guimar„es, and J.A. Ramírez. Pareto cone ϵ-dominance: Improving convergence and diversity in multiobjective evolutionary algorithms. In R.H.C. Takahashi, K. Deb, E.. Wanner, and S. Greco, editors, *Evolutionary Multi-Criterion Optimization*, volume 6576 of *Lecture Notes in Computer Science*, pages 76–90. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-19892-2. doi: 10.1007/978-3-642-19893-9_6. URL http://dx.doi.org/10.1007/978-3-642-19893-9_6.

R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

V. Belton and T. Stewart. *Muliple Criteria Decision Analysis: An Integrated Approach*. Kluwer Academic, Dordrecht, 2002.

D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987. ISBN 0132215810.

D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. I, 2nd Ed.* Athena Scientific, Belmont, MA, 2001a.

D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II, 2nd Ed.* Athena Scientific, Belmont, MA, 2001b.

M. Birattari, Z. Yuan, P. Balaprakash, and T. St¸tzle. F-race and iterated f-race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-02537-2. doi: 10.1007/978-3-642-02538-9_13. URL http://dx.doi.org/10.1007/978-3-642-02538-9_13.

V. Bowman and Jr. Joseph. On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives. In H. Thiriez and S. Zionts, editors, *Multiple Criteria Decision Making*, volume 130 of *Lecture Notes in Economics and Mathematical Systems*, pages 76–86. Springer Berlin Heidelberg, 1976. ISBN 978-3-540-07794-7. doi: 10.1007/978-3-642-87563-2_5. URL http://dx.doi.org/10.1007/978-3-642-87563-2_5.

S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed bandits. *CoRR*, abs/1001.4475, 2010.

A. Castelletti, G. Corani, A.E. Rizzoli, R. Soncini, and S.E. Weber. Reinforcement learning in the operational management of a water system. In *IFAC Workshop on Modeling and Control in Environmental Issues*, pages 325–330, 2002.

A. Castelletti, S. Galelli, M. Restelli, and R. Soncini-Sessa. Tree-based reinforcement learning for optimal water reservoir operation. *Water Resources Research*, 46, 2010. doi: 10.1029/2009WR008898.

A. Castelletti, F. Pianosi, and M. Restelli. Multi-objective fitted q-iteration: Pareto frontier approximation in one single run. In *Networking, Sensing and Control (ICNSC), 2011 IEEE International Conference on*, pages 260–265, April 2011. doi: 10.1109/ICNSC.2011.5874921.

A. Castelletti, F. Pianosi, and M Restelli. Tree-based fitted q-iteration for multi-objective markov decision problems. In *Proceedings International Joint Conference on Neural Networks (IJCNN 2012)*, 2012.

C.A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999. ISSN 0219-1377. doi: 10.1007/BF03325101. URL http://dx.doi.org/10.1007/BF03325101.

C.A. Coello Coello, G.B. Lamont, and D.A.V. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387332545.

E. Cope. Regret and convergence bounds for immediate-reward reinforcement learning with continuous action spaces. *IEEE Transactions on Automatic Control*, 54(6):1243–1253, 2009.

R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, CG'06, pages 72–83, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75537-3, 978-3-540-75537-1. URL http://dl.acm.org/citation.cfm?id=1777826.1777833.

K. Dächert, J. Gorski, and K. Klamroth. An augmented weighted tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems. *Comput. Oper. Res.*, 39(12):2929–2943, December 2012. ISSN 0305-0548. doi: 10.1016/j.cor.2012.02.021. URL http://dx.doi.org/10.1016/j.cor.2012.02.021.

I. Das and J. E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural and Multidisciplinary Optimization*, 14:63–69, 1997. ISSN 1615-147X.

I. Das and J. E. Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM J. on Optimization*, 8(3):631–657, March 1998. ISSN 1052-6234. doi: 10.1137/S1052623496307510. URL http://dx.doi.org/10.1137/S1052623496307510.

K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *Evolutionary Computation, IEEE Transactions on*, 18(4):577–601, Aug 2014. ISSN 1089-778X. doi: 10.1109/TEVC.2013.2281535.

K. Deb, J. Sundar, N. Udaya Bhaskara Rao, and S. Chaudhuri. Reference point based multi-objective optimization using evolutionary algorithms. In *International Journal of Computational Intelligence Research*, pages 635–642. Springer-Verlag, 2006.

B. Depraetere, G. Pinte, and J. Swevers. Iterative optimization of the filling phase of wet clutches. In *The 11th International Workshop on Advanced Motion Control (AMC 2010)*, pages 94–99. IEEE, 2010. ISBN 978-1-4244-6668-9.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Improving the anytime behavior of two-phase local search. *Annals of Mathematics and Artificial Intelligence*, 61(2):125–154, 2011. ISSN 1012-2443. doi: 10.1007/s10472-011-9235-0. URL http://dx.doi.org/10.1007/s10472-011-9235-0.

N. Dunford, J.T. Schwartz, W.G. Bade, and R.G. Bartle. *Linear Operators: General theory. Part. I.* Linear Operators. Interscience Publishers, 1988. ISBN 9780470226056.

M. Ehrgott. *Multicriteria Optimization*. Lectures notes in economics and mathematical systems. Springer, 2005. ISBN 9783540213987.

M. Ehrgott and X. Gandibleux. *Multiple Criteria Optimization. State of the art annotated bibliographic surveys*. Kluwer Academic, Dordrecht, 2002.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, April 2005.

L. Esterle, P.R. Lewis, M. Bogdanski, B. Rinner, and Xin Yao. A Socio-Economic Approach to Online Vision Graph Generation and Handover in Distributed Smart Camera Networks. In *Proceedings of the Fifth ACM/IEEE International Conference on Distributed Smart Cameras*, pages 1–6. IEEE Press, 2011.

L. Esterle, P.R. Lewis, H. Caine, X. Yao, and B. Rinner. CamSim: A distributed smart camera network simulator. In *Proceedings of the 7th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops (SASOW)*. IEEE Press, 2013. In press.

L. Esterle, P.R. Lewis, X. Yao, and B. Rinner. Socio-economic vision graph generation and handover in distributed smart camera networks. *ACM Transactions on Sensor Networks*, 10(2), 2014.

J. Figueira, S. Greco, and M. Ehrgott. *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer Verlag, Boston, Dordrecht, London, 2005.

C. M. Fonseca and P.J. Fleming. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-299-2. URL http://dl.acm.org/citation.cfm?id=645513.657757.

T. Furmston and D. Barber. A unifying perspective of parametric policy search methods for markov decision processes. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2717–2725. Curran Associates, Inc., 2012.

V. Gabillon, M. Ghavamzadeh, and B. Scherrer. Approximate dynamic programming finally performs well in the game of tetris. In C.J.C. Burgesv, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 1754–1762. Curran Associates, Inc., 2013.

Z. Gábor., Z. Kalmár, and C. Szepesvári. Multi-criteria reinforcement learning. In J.W. Shavlik, editor, *ICML*, pages 197–205. Morgan Kaufmann, 1998. ISBN 1-55860-556-8.

P. Geibel. Reinforcement learning for mdps with constraints. In J. F¸rnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pages 646–653. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-45375-8. doi: 10.1007/11871842_63. URL http://dx.doi.org/10.1007/11871842_63.

C.K. Goh and K.C. Tan. *Evolutionary Multi-objective Optimization in Uncertain Environments: Issues and Algorithms*. Studies in Computational Intelligence. Springer, 2009. ISBN 9783540959755.

J.-M. Gorce, R. Zhang, K. Jaffrès-Runser, and C. Goursaud. Energy, latency and capacity trade-offs in wireless multi-hop networks. In *Proceedings of the IEEE 21st International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2010*, pages 2757–2762. IEEE, 2010.

G.N.A. Hassan. *Multiobjective Genetic Programming for Financial Portfolio Management in Dynamic Environments*. University College London (University of London), 2010.

U. Hege and D. Bergemann. The Financing of Innovation: Learning and Stopping. *RAND Journal of Economics*, Vol.36,n°4:pp.719–752, 2005.

F. Hernandez-del Olmo, F. H. Llanes, and E. Gaudioso. An emergent approach for the control of wastewater treatment plants by means of reinforcement learning techniques. *Expert Syst. Appl.*, 39(3):2355–2360, 2012.

John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0262082136.

S. Huband, P. Hingston, L. Barone, and L. While. A review of multiobjective test problems and a scalable test problem toolkit. *Trans. Evol. Comp*, 10(5):477–506, October 2006. ISSN 1089-778X. doi: 10.1109/TEVC.2005.861417. URL http://dx.doi.org/10.1109/TEVC.2005.861417.

M. Humphrys. *Action selection methods using reinforcement learning*. PhD thesis, University of Cambridge, Computer Laboratory, June 1997. URL http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-426.ps.gz.

R. Issabekov and P. Vamplew. An empirical comparison of two common multiobjective reinforcement learning algorithms. In *Proceedings of the 25th Australasian Joint Conference on Advances in Artificial Intelligence*, AI'12, pages 626–636, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-35100-6. doi: 10.1007/978-3-642-35101-3_53. URL http://dx.doi.org/10.1007/978-3-642-35101-3_53.

H. Jain and K. Deb. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: Handling constraints and extending to an adaptive approach. *Evolutionary Computation, IEEE Transactions on*, 18(4):602–622, Aug 2014. ISSN 1089-778X. doi: 10.1109/TEVC.2013.2281534.

A. Jaszkiewicz and R. Słowiński. The light beam search over a non-dominated surface of a multiple-objective programming problem. In G.H. Tzeng, H.F. Wang, U.P. Wen, and P.L. Yu, editors, *Multiple Criteria Decision Making*, pages 87–99. Springer New York, 1994. ISBN 978-1-4612-7626-5. doi: 10.1007/978-1-4612-2666-6_10. URL http://dx.doi.org/10.1007/978-1-4612-2666-6_10.

A. Jaszkiewicz and R. Słowiński. The light beam search - outranking based interactive procedure for multiple-objective mathematical programming. In PanosM. Pardalos, Yannis Siskos, and Constantin Zopounidis, editors, *Advances in Multicriteria Analysis*, volume 5 of *Nonconvex Optimization and Its Applications*, pages 129–146. Springer US, 1995. ISBN 978-1-4419-4748-2. doi: 10.1007/978-1-4757-2383-0_9. URL http://dx.doi.org/10.1007/978-1-4757-2383-0_9.

N.R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, January 1998. ISSN 1387-2532. doi: 10.1023/A:1010090405266. URL http://dx.doi.org/10.1023/A:1010090405266.

N. Jozefowiez, F. Semet, and E.-G. Talbi. Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293 – 309, 2008. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2007.05.055.

L. Kaelbling, M. Littman, and A. Cassandra. Partially observable markov decision processes for artificial intelligence. In Ipke Wachsmuth, Claus-Rainer Rollinger, and Wilfried Brauer, editors, *KI-95: Advances in Artificial Intelligence*, volume 981 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 1995. ISBN 978-3-540-60343-6. doi: 10.1007/3-540-60343-3_22. URL http://dx.doi.org/10.1007/3-540-60343-3_22.

L. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

D. Kalyanmoy and K. Abhay. Light beam search based multi-objective optimization using evolutionary algorithms. In *IEEE Congress on Evolutionary Computation*, pages 2125–2132, 2007. doi: 10.1109/CEC.2007.4424735.

H. Kazuyuki, Y. Manabu, and M. Taketoshi. Parallel reinforcement learning for weighted multi-criteria model with adaptive margin. *Cognitive Neurodynamics*, 3(1):17–24, 2009. ISSN 1871-4080. doi: 10.1007/s11571-008-9066-9. URL http://dx.doi.org/10.1007/s11571-008-9066-9.

I.Y. Kim and O.L. de Weck. Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and Multidisciplinary Optimization*, 29(2):149–158, 2005. ISSN 1615-147X. doi: 10.1007/s00158-004-0465-1. URL http://dx.doi.org/10.1007/s00158-004-0465-1.

K. Klamroth and T. Jørgen. Constrained optimization using multiple objective programming. *Journal of Global Optimization*, 37(3):325–355, 2007. ISSN 0925-5001. doi: 10.1007/s10898-006-9052-x. URL http://dx.doi.org/10.1007/s10898-006-9052-x.

J. Knowles and D. Corne. On metrics for comparing nondominated sets. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 711–716, May 2002. doi: 10.1109/CEC.2002.1007013.

J. Knowles, R. Watson, and D. Corne. Reducing local optima in single-objective problems by multi-objectivization. In *Evolutionary Multi-Criterion Optimization*, pages 269–283. Springer, 2001.

M. Laumanns, L. Thiele, K. Deb, and E. Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evol. Comput.*, 10(3):263–282, September 2002. ISSN 1063-6560. doi: 10.1162/106365602760234108. URL http://dx.doi.org/10.1162/106365602760234108.

P.R. Lewis, L. Esterle, A. Chandra, B. Rinner, and X. Yao. Learning to be different: Heterogeneity and efficiency in distributed smart camera networks. In *Proceedings of the 7th IEEE Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 209–218. IEEE Press, 2013.

D. J. Lizotte, M. Bowling, and S. A. Murphy. Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML)*, pages 695–702, 2010.

D. J. Lizotte, M. Bowling, and S. A. Murphy. Linear fitted-q iteration with multiple reward functions. *Journal of Machine Learning Research*, 13:3253–3295, 2012.

T. Lust and J. Teghem. Two-phase pareto local search for the biobjective traveling salesman problem. *Journal of Heuristics*, 16(3):475–510, 2010. ISSN 1381-1231. doi: 10.1007/s10732-009-9103-9. URL http://dx.doi.org/10.1007/s10732-009-9103-9.

S. Mannor and N. Shimkin. The steering approach for multi-criteria reinforcement learning. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS*, pages 1563–1570. MIT Press, 2001.

S Mannor and N. Shimkin. The steering approach for multi-criteria reinforcement learning. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1563–1570. MIT Press, 2002.

R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004. ISSN 1615-147X. doi: 10.1007/s00158-003-0368-6. URL http://dx.doi.org/10.1007/s00158-003-0368-6.

A. Messac and C. Mattson. Generating well-distributed sets of pareto points for engineering design using physical programming. *Optimization and Engineering*, 3(4):431–450, 2002. ISSN 1389-4420. doi: 10.1023/A:1021179727569. URL http://dx.doi.org/10.1023/A%3A1021179727569.

K. Miettinen. *Nonlinear Multiobjective Optimization*. International Series in Operations Research & Management Science. Springer US, 1999. ISBN 9780792382782.

H. Nakayama, Y. Yun, and M. Yoon. *Sequential Approximate Multiobjective Optimization Using Computational Intelligence*. Vector Optimization. Springer, 2009. ISBN 9783540889106.

S. Natarajan and P. Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 601–608, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: 10.1145/1102351.1102427. URL http://doi.acm.org/10.1145/1102351.1102427.

A. Nowé, P. Vrancx, and Y-M. De Hauwere. Game theory and multi-agent reinforcement learning. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State-Of-The-Art*, volume 12 of *Adaptation, Learning, and Optimization*, pages 441–470. Springer, Heidelberg, Germany, 2012. doi: 10.1007/978-3-642-27645-3_3.

S. Pandey, D. Chakrabarti, and D. Agarwal. Multi-armed bandit problems with dependent arms. In *Proceedings of the 24th International Conference on Machine Learning*, 2007.

V. Pareto. *Cours d'Economie Politique*. Droz, Genève, 1896.

J. Perez, C. Germain Renaud, B. Kégl, and C. Loomis. Responsive Elastic Computing. In *2009 ACM/IEEE Conference on International Conference on Autonomic Computing*, pages 55–64, Barcelone, Spain, June 2009. ACM. doi: 10.1145/1555301.1555311.

P. Perny and P. Weng. On finding compromise solutions in multiobjective markov decision processes. In H. Coelho, R. Studer, and M. Wooldridge, editors, *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 969–970. IOS Press, 2010. ISBN 978-1-60750-605-8.

G. Pinte, J. Stoev, W. Symens, A. Dutta, Y. Zhong, B. Wyns, R. De Keyser, B. Depraetere, J. Swevers, M. Gagliolo, and A. Nowé. Learning strategies for wet clutch control. In *15th International Conference on System Theory, Control and Computing — ICSTCC 2011*, pages 467–474. IEEE, 2011. ISBN 978-973-621-322-9.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 952.

D. M. Roijers, P. Vamplew, S. Whiteson, and R. Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res. (JAIR)*, 48:67–113, 2013.

M. Sakawa and K. Kato. Interactive decision-making for multiobjective linear fractional programming problem with block angular structure involving fuzzy numbers. *Fuzzy Sets and Systems*, 97(1):19–32, 1998.

M. Sakawa and T. Shibano. An interactive fuzzy satisficing method for multiobjective 0-1 programming problems with fuzzy numbers through genetic algorithms with double strings. *European Journal of Operational Research*, 107(3):564–574, 1998.

J.D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc. ISBN 0-8058-0426-9. URL http://dl.acm.org/citation.cfm?id=645511.657079.

D. Schultz and S. Schultz. *A History of Modern Psychology*. Cengage Learning, 2011. ISBN 9781111344979.

W.M. Spears. *Evolutionary Algorithms: The Role of Mutation and Recombination*. Natural Computing Series. Springer, 2000. ISBN 9783540669500.

R.. Steuer and E. Choo. An interactive weighted tchebycheff procedure for multiple objective programming. *Mathematical Programming*, 26(3):326–344, 1983. ISSN 0025-5610. doi: 10.1007/BF02591870. URL http://dx.doi.org/10.1007/BF02591870.

T.J. Stewart. A critical survey on the status of multiple criteria decision making theory and practice. *Omega*, 20(5?6):569 – 586, 1992. ISSN 0305-0483. doi: http://dx.doi.org/10.1016/0305-0483(92)90003-P.

R. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991. ISSN 0163-5719. doi: 10.1145/122344.122377. URL http://doi.acm.org/10.1145/122344.122377.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. Mit Press, 1998. ISBN 9780262193986.

G. Tesauro, R. Das, H. Chan, J.O. Kephart, D.W. Levine, F. Rawson, and C. Lefurgy. Managing power consumption and performance of computing systems using reinforcement learning. In J.C. Platt, D. Koller, Y. Singer, and S.T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1497–1504. Curran Associates, Inc., 2008.

E.L. Thorndike. *Animal Intelligence: Experimental Studies*. The animal behaviour series. Macmillan, 1911.

J. N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Journal of Machine Learning*, 16(3):185–202, 1994.

G.H. Tzeng and J.J. Huang. *Multiple Attribute Decision Making: Methods and Applications*. A Chapman & Hall book. Taylor & Francis, 2011. ISBN 9781439861578.

E. Uchibe and K. Doya. Constrained reinforcement learning from intrinsic and extrinsic rewards. In *Development and Learning, 2007. ICDL 2007. IEEE 6th International Conference on*, pages 163–168, July 2007. doi: 10.1109/DEVLRN.2007.4354030.

P. Vamplew, J. Yearwood, R. Dazeley, and A. Berry. On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. In *Proceedings of the 21st Australasian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, AI '08, pages 372–378, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89377-6.

P. Vamplew, R. Dazeley, E. Barker, and A. Kelarev. Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks. In Ann E. Nicholson and Xiaodong Li, editors, *Australasian Conference on Artificial Intelligence*, volume 5866 of *Lecture Notes in Computer Science*, pages 340–349. Springer, 2009. ISBN 978-3-642-10438-1.

P. Vamplew, R. Dazeley, A. Berry, R. Issabekov, and E. Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84 (1-2):51–80, 2010.

K. Van Moffaert and A. Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *Journal of Machine Learning Research*, 15:3483–3512, 2014.

K. Van Moffaert, Brys, and A. Ann Nowé. Efficient weight space search in multi-objective reinforcement learning. Technical Report AI-TR-14-69, Computational Modeling Lab, Department of Computer Science, Vrije Universiteit Brussel, Belgium, 2013.

K. Van Moffaert, M. M. Drugan, and A. Nowé. Hypervolume-based multi-objective reinforcement learning. In R. Purshouse, P. Fleming, C. Fonseca, S. Greco, and J. Shaw, editors, *Evolutionary Multi-Criterion Optimization*, volume 7811 of *Lecture Notes in Computer Science*, pages 352–366. Springer Berlin Heidelberg, 2013a. ISBN 978-3-642-37139-4.

K. Van Moffaert, M. M. Drugan, and A. Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In *ADPRL*, pages 191–199. IEEE, 2013b. ISBN 978-1-4673-5925-2.

K. Van Moffaert, M. M. Drugan, and A. Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In K. Hindriks, M. de Weerdt, B. van Riemsdijk, and M. Warnier, editors, *Proceedings of the 25th Benelux Conference on Artificial Intelligence, BNAIC 2013, Delft, the Netherlands*, pages 360–361, Delft, The Netherlands, November 2013c. Delft University of Technology.

K. Van Moffaert, T. Brys, A. Chandra, L. Esterle, P.R. Lewis, and A. Nowé. A novel adaptive weight selection algorithm for multi-objective multi-agent reinforcement learning. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 2306–2314, July 2014a. doi: 10.1109/IJCNN.2014.6889637.

K. Van Moffaert, K. Van Vaerenbergh, P. Vrancx, and A. Nowé. Multi-objective $\mathcal{X}$-armed bandits. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 2331–2338, July 2014b. doi: 10.1109/IJCNN.2014.6889753.

K. Van Moffaert, T. Brys, and A. Nowé. Risk-sensitivity through multi-objective reinforcement learning. In *2015 Congress of Evolutionary Computation*, pages 2331–2338, May 2015. doi: 10.1109/IJCNN.2014.6889753.

K. Van Vaerenbergh, P. Vrancx, Y.-M. De Hauwere, A. Nowé, E. Hostens, and C. Lauwerys. Tuning hydrostatic two-output drive-train controllers using reinforcement learning. *Mechatronics*, 24(8):975 – 985, 2014. ISSN 0957-4158. doi: http://dx.doi.org/10.1016/j.mechatronics.2014.07.005.

D. A. Van Veldhuizen and G. B. Lamont. Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1998.

W. Wang and M. Sebag. Multi-objective monte-carlo tree search. In S. Hoi and W.L. Buntine, editors, *Proceedings of the 4th Asian Conference on Machine Learning, ACML 2012, Singapore, Singapore, November 4-6, 2012*, volume 25 of *JMLR Proceedings*, pages 507–522. JMLR.org, 2012. URL http://jmlr.org/proceedings/papers/v25/.

W. Wang and M. Sebag. Hypervolume indicator and dominance reward based multi-objective monte-carlo tree search. *Machine Learning*, 92(2-3):403–429, 2013. ISSN 0885-6125. doi: 10.1007/s10994-013-5369-0.

C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, England, 1989.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992. doi: 10.1023/A:1022676722315.

G. Weiß, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1999.

D. J. White. Multi-objective infinite-horizon discounted Markov decision processes. *Journal of Mathematical Analysis and Applications*, 89(2), 1982.

M. A. Wiering and E. D. de Jong. Computing Optimal Stationary Policies for Multi-Objective Markov Decision Processes. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 158–165. IEEE, April 2007. ISBN 1-4244-0706-0.

P. Winkler. Random orders. *Order*, 1(4):317–331, 1985. ISSN 0167-8094. doi: 10.1007/BF00582738. URL http://dx.doi.org/10.1007/BF00582738.

G. Xu, Y. Zong, and Z. Yang. *Applied Data Mining*. CRC Press, 2013. ISBN 9781466585843.

J. Yaochu. *Advanced fuzzy systems design and applications.*, volume 112 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag, 2003. ISBN 3-7908-1537-3.

L. Yliniemi and K. Tumer. PaCcET: An objective space transformation to iteratively convexify the pareto front. In *The Tenth International Conference on Simulated Evolution And Learning (SEAL 2014)*, Dunedin, New Zealand, December 2014.

Y. Zhong, B. Wyns, R. De Keyser, G. Pinte, and J. Stoev. Implementation of genetic based learning classifier system on wet clutch system. In *30th Benelux Meeting on Systems and Control — Book of Abstracts*, page 124, Gent, Belgium, 2011. Universiteit Gent. ISBN 978-90-9026089-1.

S. Zionts. Mcdm? if not a roman numeral, then what? *Interfaces*, 9(4):94–101, 1979. doi: 10.1287/inte.9.4.94. URL http://dx.doi.org/10.1287/inte.9.4.94.

E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003a.

E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, and V.G. da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, April 2003b. ISSN 1089-778X. doi: 10.1109/TEVC.2003.810758.

Index

T

V

W