University of Luxembourg

DISSERTATION

Multi-Objective Reinforcement Learning

Presented on the 25th of June 2024 in Luxembourg to obtain the degree of

"DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG EN INFORMATIQUE"

by

Florian Felten

Jury members:

Chairman:	Prof. Dr. Pascal Bouvry	University of Luxembourg
Vice-chairman:	Prof. Dr. El-Ghazali Talbi	Université de Lille
Supervisor:	Dr. Grégoire Danoy	University of Luxembourg
Member:	Prof. Dr. Peter Vamplew	Federation University Australia
Member:	Dr. Michal Valko	Meta AI

July 2024

Abstract

The recent surge in artificial intelligence (AI) agents assisting us in daily tasks suggests that these agents possess the ability to comprehend key aspects of our environment, thereby facilitating better decision-making. Presently, this understanding is predominantly acquired through data-driven learning methods. Notably, reinforcement learning (RL) stands out as a natural framework for agents to acquire behaviors by interacting with their environment and learning from feedback. However, despite the effectiveness of RL in training agents to optimize a single objective, such as minimizing cost or maximizing performance, it overlooks the inherent complexity of decision-making in real-world scenarios where multiple objectives may need to be considered simultaneously. Indeed, an essential aspect that remains understudied is the human tendency to make compromises in various situations, influenced by values, circumstances, or mood. This limitation underscores the need for advancements in AI methodologies to address the nuanced trade-offs inherent to human decision-making. Thus, this work aims to explore the extension of RL principles into multi-objective settings, where agents can learn behaviors that balance competing objectives, thereby enabling more adaptable and personalized AI systems.

In the first part of this thesis, we explore the domain of multi-objective reinforcement learning (MORL), a recent technique aimed at enabling AI agents to acquire diverse behaviors associated with different trade-offs from multiple feedback signals. While MORL is relatively recent, works in this field often rely on existing knowledge coming from older fields such as multi-objective optimization (MOO) and RL. Our initial contribution involves a comprehensive analysis of the relationships between RL, MOO, and MORL. This examination culminates in the development of a taxonomy for categorizing MORL algorithms, drawing on concepts derived from preceding fields. Building upon this foundational understanding, we proceed to investigate the feasibility of leveraging techniques from MOO and RL to enhance MORL methodologies. This exploration yields several contributions. Among these, we introduce the utilization of metaheuristics to address the exploration-exploitation dilemma in MORL. Additionally, we introduce a versatile framework rooted in the derived taxonomy, facilitating the creation of novel MORL algorithms based on techniques coming from MOO and RL. Furthermore, our efforts extend towards improving the scientific rigor and practical applicability of MORL in real-world scenarios. To this end, we introduce methods and a suite of open-source tools that have become the standard in MORL.

Many real-world situations also involve collaboration among multiple agents to accomplish tasks efficiently. Therefore, the second part of this thesis transitions to settings involving multiple agents, leading to the nascent field of multi-objective multi-agent reinforcement learning (MOMARL). In this domain, as an initial contribution, we release a comprehensive set of open-source utilities aimed to accelerate and establish a robust foundation for research within this evolving domain. Furthermore, we perform an initial study exploring the transferability of knowledge and methodologies from both MORL and multi-agent RL to the MOMARL settings. Finally, we validate our approach in a real-world application. Specifically, we aim to automatically learn the coordination of multiple drones having different objectives, harnessing the MOMARL framework to orchestrate their actions effectively. This empirical validation serves as evidence of the viability and versatility of the proposed methodologies in addressing complex real-world challenges.

Keywords: reinforcement learning, multi-objective, multi-agent

Acknowledgements

To begin with, I want to extend my heartfelt thanks to my advisor, Dr. Grégoire DANOY, for his unwavering support, encouragement, expertise, and humor. I couldn't have wished for a better advisor for my studies and a friend to share a drink with.

I am also grateful to the other members of my thesis committee: Dr. El-Ghazali TALBI, for giving guidance on every research topic and raising valuable points in various discussions, and Dr. Pascal BOUVRY, for his valuable feedback, and hosting me in his research lab. Moreover, I would like to thank Dr. Peter VAMPLEW and Dr. Michal VALKO for taking the time to read this manuscript and their valuable insights.

A special thanks to my fellow labmates from PCOG for the stimulating discussions, for the love of coffee, and for all the fun we have had in the last few years. A special mention to Pierre, with whom I really connected despite his questionable research choices.

I am deeply indebted to my family for their unwavering support and encouragement throughout my years of study and through the process of researching and writing this thesis. A special thanks to my mother, Bernadette without whom my brother and I would not be the persons we are today.

Lastly, I would like to thank my friends for always believing in me and cheering me up, even when I was at my lowest. Nikos, Maxou, le Ritch, Francis, Gigi, Mel, Yousa, Guibs, Tutur, Burio, Edou: merci et chapeau.

Thank you all.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

RL	**R**einforcement **L**earning
MDP	**M**arkov **D**ecision **P**rocess
DNN	**D**eep **N**eural **N**etwork
PER	**P**rioritized **E**xperience **R**eplay
HER	**H**indsight **E**xperience **R**eplay
DQN	**D**eep **Q**-**N**etwork
PPO	**P**roximal **P**olicy **O**ptimization
SAC	**S**oft **A**ctor-**C**ritic
MOO	**M**ulti-**O**bjective **O**ptimization
DM	**D**ecision **M**aker
PF	**P**areto **F**ront
PS	**P**areto **S**et
IGD	**I**nverted **G**enerational **D**istance
MUL	**M**aximum **U**tility **L**oss
MOP	**M**ulti-objective **O**ptimization **P**roblem
SOP	**S**ingle-objective **O**ptimization **P**roblem
MOO/D	**M**ulti-**O**bjective **O**ptimization based on **D**ecomposition
PSA	**P**areto **S**imulated **A**nnealing
EA	**E**volutionary **A**lgorithm
NSGA	**N**on-dominated **S**orting **G**enetic **A**lgorithm
IBEA	**I**ndicator-**B**ased **E**volutionary **A**lgorithm
MORL	**M**ulti-**O**bjective **R**einforcement **L**earning
MOMDP	**M**ulti-**O**bjective **M**arkov **D**ecision **P**rocess
MORL/D	**M**ulti-**O**bjective **R**einforcement **L**earning based on **D**ecomposition
FIFO	**F**irst-**I**n-**F**irst-**O**ut
SER	**S**calarized **E**xpected **R**eturn
ESR	**E**xpected **S**calarized **R**eturn
EED	**E**xploration-**E**xploitation **D**ilemma
ORLB	**O**pen **R**einforcement **L**earning **B**enchmark
MARL	**M**ulti-**A**gent **R**einforcement **L**earning
W&B	**W**eights **&** **B**iases
CLI	**C**ommand-**L**ine **I**nterface
HPO	**H**yper**p**arameter **O**ptimization
BO	**B**ayesian **O**ptimization
MOMARL	**M**ulti-**O**bjective **M**ulti-**A**gent **R**einforcement **L**earning
MOMMDP	**M**ulti-**O**bjective **M**ulti-**A**gent **M**arkov **D**ecision **P**rocess

MODec-POMDP	**M**ulti-**O**bjective **D**ecentralized **P**artially **O**bservable **M**arkov **D**ecision **P**rocess
CTDE	**C**entralized **T**raining **D**ecentralized **E**xecution
IQL	**I**ndependent **Q**-**L**earning
MADDPG	**M**ulti-**A**gent **D**eep **D**eterministic **P**olicy **G**radient
MAPPO	**M**ulti-**A**gent **P**roximal **P**olicy **O**ptimization
MOMAAD	**M**ulti-**O**bjective **M**ulti-**A**gent Reinforcement Learning based on **A**ccelerated **D**ecomposition

Symbols

Reinforcement learning	
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
r	Reward function
p	Transition function
μ_0	Distribution over initial states
γ	Discount factor
π	Policy
π^*	Optimal policy
$v^\pi(s)$	Value function of a policy π given an initial state s
$q^\pi(s,a)$	State-action value function of a policy π given a state s and an action a
v^π	Value of a policy π, regardless of the initial state
$\tilde{q}(s,a)$	Estimation of the state-action value
θ	Parameters (optimized in function approximations)
Θ	Parameter space
Multi-objective optimization	
$\mathbf{x}$	Decision variables
Ω	Decision (or search) space
$\mathbf{f}$	Multi-objective evaluation function
$\mathcal{PS}$	Pareto Set
$\mathcal{F}(\mathcal{PS})$	Pareto Front of a given Pareto Set
$\tilde{\mathcal{F}}$	Approximated Pareto front
$\mathcal{Z}$	Reference (optimal) Pareto front
$\mathbf{z}$	Reference point (used in hypervolume or scalarization functions)
Multi-objective RL	
$\mathbf{r}$	Vectorial reward signal
$\mathbf{v}^\pi$	Vectorial value function of a policy π
$\mathcal{Q}(s,a)$	Q-set of a given state s and action a
Multi-objective multi-agent RL	
$\mathcal{S}^n$	Joint state spaces of all agents
$\mathcal{A}^n$	Joint action spaces of all agents
$\boldsymbol{\pi}$	Joint policy
$\mathbf{v}^{\boldsymbol{\pi}}$	Vectorial value function of a joint policy
Hyperparameter optimization	
$\boldsymbol{\psi}$	Hyperparameters
Ψ	Hyperparameter space

σ	Random seed
Σ_{search}	Set of random seeds employed while searching for hyperparameters
$\Sigma_{\text{validation}}$	Set of random seeds employed while validating final performance

Chapter 1

Introduction

Contents

1.1 Motivation

In recent years, the domain of artificial intelligence (AI) has experienced remarkable advancements, fundamentally reshaping various aspects of our daily lives. In less than a decade, the strides made in AI have been extraordinary, ranging from achieving mastery in video games [Mnih et al., 2015] to offering invaluable support in tasks such as writing and idea generation (*e.g.*, OpenAI's ChatGPT). As a result, AI has emerged as a focal point of interest in today's economy, increasingly becoming an integral part of modern society. While the broad realm of AI includes areas such as machine learning (ML), optimization, and reasoning [Russell and Norvig, 2010], recent breakthroughs in AI have largely been driven by ML methods, evidenced by significant achievements like protein folding [Jumper et al., 2021]. Consequently, ML has attracted considerably more attention than other AI fields in recent years. Breakthroughs in computational hardware design and production, the abundance and accessibility of training data, and the continuous evolution of ML algorithms have been instrumental in propelling these advancements.

At the forefront of recent technical advancements in ML lies reinforcement learning (RL) [Sutton and Barto, 2018]. In this paradigm, an AI agent learns from experiences gathered by sequentially interacting with an environment. These experiences, typically synthetically generated from a simulator mirroring real-world scenarios, allow the agent to receive feedback signals, which can be positive, negative, or neutral. Based on these experiences, the agent learns to take actions leading to favorable outcomes, akin to how humans and animals learn from rewards.

Despite being in existence for decades, RL has only recently garnered significant attention from the research and industrial community. This is largely due to its breakthrough success in outplaying professional human players in the game of Go [Silver et al., 2016]. Consequently, in recent years, RL has been applied to various challenging applications, including large language models [Ouyang et al., 2022], drone racing [Kaufmann et al., 2023], and the control of tokamak plasmas in nuclear fusion reactors [Degrave et al., 2022, Tracey et al., 2024].

Balancing Among Multiple Objectives

Regardless of the significant recent successes achieved in various domains, several challenges persist. Contrary to games where the objective is typically binary (win or lose), real-world scenarios often involve balancing competing objectives, such as optimizing performance while minimizing resource consumption or maximizing safety [Vamplew et al., 2022]. In this context, the thesis first explores the domain of multi-objective reinforcement learning (MORL) [Roijers and Whiteson, 2017], which provides a means to automatically learn agents' behaviors (also called policy) that achieve a balance among multiple objectives.

Over the past decade, MORL has been the subject of growing attention from the research community, though it still constitutes a small segment of the broader RL research field. Indeed, although most real-world problems involve making a compromise among various objectives, a large portion of the RL research community remains unaware of the MORL field. In practice, MO problems are often overlooked and objectives are condensed into a single objective using a *scalarization function* with pre-defined importance among objectives. However, this naive solution can result in undesirable or less-than-optimal outcomes [Hayes et al., 2022].

Systems Involving More Than One Agent

Furthermore, some applications involve coordinating the actions of multiple collaborative agents in dynamic environments, *e.g.*, learning to control a swarm of drones [Bayındır, 2016, Hüttenrauch et al., 2019]. Thus, the thesis also explores concepts of multi-agent reinforcement learning (MARL) [Foerster, 2018], enabling learning in environments populated by multiple interacting agents.

Combining both aspects of MORL and MARL, the field of multi-objective multi-agent RL (MOMARL) has recently been defined [Rădulescu et al., 2020]. Here, agents aim to learn behaviors associated with various compromises while taking account of each other. While the development of MOMARL methods is justified by real-world needs, little attention has been directed towards this area of research until now [Hayes et al., 2022].

To continue the discussion of our multi-objective and multi-agent settings, we present two motivating examples inspired by modern real-life applications.

Motivating Examples

We introduce two illustrative examples to underscore the importance of multi-objective settings in both single and multi-agent systems.

Example 1.1 (A Robot Learning to Run)**.** *Consider an AI agent tasked with learning to make a robot run. The agent receives information on the robot's current state, including rotor angles and velocities, as well as parts' positions and velocities. At each time step, the agent determines torque distribution to the motors to maximize the robot's velocity. However, focusing solely on velocity may lead to inefficient actions, quickly draining the robot's battery. By introducing an additional objective such as saving energy, the agent can explore diverse behaviors, balancing energy efficiency with velocity.*

Example 1.2 (A Team of Drones Observing an Object)**.** *Now, let us imagine a scenario where multiple AI agents are tasked with directing drones to observe a particular object, a situation frequently encountered in infrastructure inspection tasks. Each agent receives information about the GPS coordinates of all drones involved and the progress of the object's scanning process. These agents are responsible for determining the movements of the drones, aiming to facilitate efficient scanning while simultaneously avoiding collisions. The objectives*

include maintaining proximity to the object for thorough scanning, ensuring safe distances between drones, and strategically spreading around the object to maximize coverage.

In such scenarios, traditional approaches typically attempt to determine the relative importance of each objective by assigning fixed coefficients to them, combining them, and then running the learning process to observe resulting behaviors in a trial-and-error manner. In contrast, by treating the scenarios as multi-objective problems, agents can dynamically explore various trade-offs throughout the learning process and present these options to the user. This allows users to select from the array of compromise behaviors identified by the algorithm before execution in real-world conditions. This flexibility in the decision-making process stands as a key motivation behind the methods discussed throughout this thesis.

Aim and Objectives

While MORL and MOMARL offer various advantages compared to traditional methods, these fields are relatively recent. MORL emerged less than 20 years ago [Roijers et al., 2013], and MOMARL has seen little to no existing work [Rădulescu et al., 2020]. Despite a recent surge of interest in MORL, it still lags far behind RL in terms of research attention.

Now, many recent papers in MORL present novel methods inspired by other fields such as multi-objective optimization (MOO) and RL. These fields have a longer history and have been the subject of extensive research over several decades. However, while recent MORL papers draw techniques from these fields, the overlaps and differences between MOO, RL, and MORL remain unclear. Similarly, questions arise about the extent to which knowledge from MORL and MARL can be applied to MOMARL. These interrogations form the basis of our two first research questions.

Moreover, despite the recent surge of interest in RL, the development of robust and well-considered tools for proper evaluation practices has not kept pace, resulting in a reproducibility crisis within the RL community. Consequently, researchers often question the reliability and reproducibility of results presented in influential papers [Agarwal et al., 2021, Patterson et al., 2023], also hindering the deployment of RL solutions in real-world scenarios. MORL inherits some of these challenges and currently lacks defined methodologies for conducting scientifically rigorous experiments. MOMARL, being in its infancy, has the opportunity to establish proper scientific methodologies early on. Our third research question aims to study the methods and metrics that are relevant for scientific rigor in MO(MA)RL research.

Finally, while much RL research focuses on simulated scenarios, deploying learned behaviors in real-world devices can provide valuable insights. Therefore, our final question aims to explore the issues arising when pushing these boundaries and deploying MOMARL methods in real-world scenarios, moving beyond simulation.

Research Questions

Based on these observations, the main research questions addressed in the manuscript are the following:

Q1: What are the commonalities and differences between the fields of MOO, RL, and MORL?

Q2: To what extent can existing methods and knowledge from older fields be applied to MO(MA)RL?

Q3: What constitutes a rigorous scientific methodology for conducting MO(MA)RL research?

Q4: What challenges hinder the deployment of MO(MA)RL methods in real-world scenarios?

1.2 Contributions

The main contributions of this work, aimed at addressing the research questions presented above, are listed below.

An MORL Taxonomy Based on MOO and RL

To address **Q1**, we conduct a comprehensive analysis of the current state of the art in MOO, RL, and MORL. Our first contribution is the introduction of a novel taxonomy allowing the classification of MORL algorithms. Drawing insights from existing works in both RL and MOO, this taxonomy identifies key decisions made when designing MORL algorithms and provides a common vocabulary for researchers in MORL, MOO, and RL. Moreover, our taxonomy allows for straightforward identification and understanding of research contributions, which can sometimes be challenging in current papers due to their complexity.

Transferring Knowledge Across Domains

To answer **Q2**, we contribute to transferring knowledge across domains in the following ways:

Firstly, we introduce novel exploration strategies for MORL by employing metaheuristics, a classic technique from MOO. In optimization, these techniques are used to guide the search process toward favorable regions within a reasonable timeframe, often sacrificing optimality guarantees for performance. Contrasting metaheuristics with commonly used ϵ-greedy strategies in MORL, we establish the superiority of metaheuristics across various environments.

Secondly, we apply MOO techniques to MORL based on decomposition. Leveraging the taxonomy proposed in our background analysis, we introduce the MORL/D framework. This modular framework eases the creation of MORL algorithms by enabling the instantiation of building blocks using techniques coming from both MOO and RL domains. When applying this framework to diverse MORL problems, we develop new algorithms and demonstrate their competitiveness against current state-of-the-art methods.

Lastly, we demonstrate the transferability of MORL and MARL methods to the MOMARL domain by deriving learning algorithms that rely on methods coming from these fields for solving MOMARL problems. Two main techniques, decomposition and solving via MARL, and centralization and solving via MORL, are identified to enable this transfer.

Tools and Methods for Reliable MO(MA)RL

To answer **Q3**, our focus shifts towards enhancing the scientific methodology in MO(MA)RL research. Indeed, before the appearance of our contributbaselineMA)RL research involved implementing benchmark environments, implementing baselines algorithms that achieve similar results as what was published in the paper introducing the method, and comparing new methods against these baselines on potentially new environments. However, due to the substantial efforts involved, comparisons often suffered from biases, as researchers were generally reluctant to invest weeks in reproducing existing results. To solve these issues, five key contributions are presented:

1. MO-Gymnasium: a standard API and set of benchmark environments for MORL research and development;

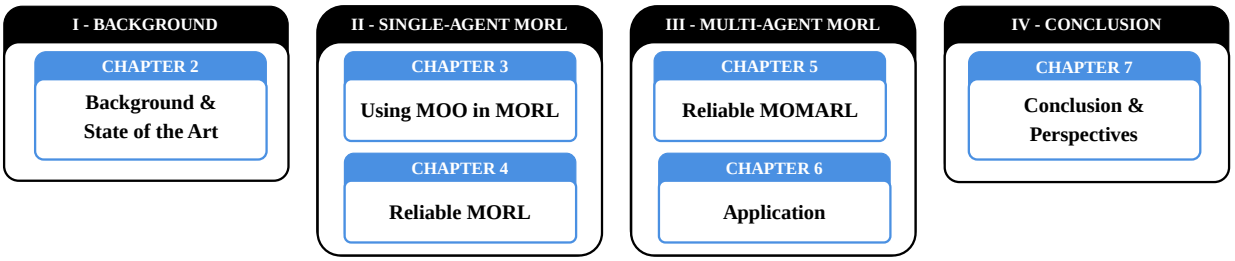


FIGURE 1.1: Organization of the dissertation.

2. MORL-Baselines: a collection of reliable implementations of MORL algorithms, compatible with MO-Gymnasium;

3. A publicly available dataset comprising results obtained from training MORL-Baselines algorithms on MO-Gymnasium environments. This dataset offers insights into algorithm performance across different environments. Moreover, it allows extracting the data directly, removing the need to re-run baselines for MORL researchers;

4. A study on the problem of optimizing hyperparameter values for MORL algorithms as well as an automated method to solve this issue. This not only simplifies the deployment of MORL solutions but also facilitates fairer comparisons between algorithms;

5. MOMAland: a standard API and set of benchmark environments for MOMARL. MOMAland also contains utilities and algorithms aimed to provide initial open-source baselines for this field of research.

These tools aim to streamline application and research in these burgeoning fields, which are steadily gaining traction in both academia and industry.

Easing the Application of MO(MA)RL

To address **Q4**, we first want to emphasize that the tools mentioned earlier also contribute to easing the applicability of these methods in real-world scenarios. Before the publication of tools like MO-Gymnasium, MORL-Baselines, and MOMAland, engineers often had limited access to open-source code and resources, resulting in significant costs in terms of human effort for the implementation of such methods.

As a second contribution to study Q4, we illustrate an application of MOMARL in real-world scenarios. Similar to Example 1.2 mentioned above, we aim to learn cooperative behaviors for multiple agents with multiple objectives, specifically focusing on controlling a swarm of drones that aim to make a formation around a potentially moving target. This task requires the drones to collaborate effectively to surround the target rather than acting selfishly. Additionally, the drones must strike a balance between proximity to the target and spacing between themselves. Importantly, while this final application showcases the control of drone swarms, the insights and methodologies developed have broader implications across a spectrum of domains and applications.

1.3 Thesis Structure

An overview of the thesis structure is depicted in Figure 1.1. The thesis starts with background information and then is followed by two parts (detailed below) tackling the diverse aspects described above, a final discussion and concluding remarks are held in the last part.

Part I: Background

Chapter 2 serves as the foundation, providing essential information, including notations, solution concepts, and the current state of the art in the fields under study. Notably, it initially describes the domains of RL and MOO, laying the groundwork for subsequent discussions. Following this, it introduces the recent field of MORL and our first contribution, the MORL/D taxonomy. After presenting MORL, this field is expanded to multi-agent settings, leading to the nascent field of multi-objective multi-agent reinforcement learning.

Part II: Single-Agent Multi-Objective Reinforcement Learning

Part II describes our contributions in the domain of single-agent MORL. In Chapter 3, we present our algorithmic contributions aimed at demonstrating the applicability of existing MOO and RL methods in MORL. Additionally, Chapter 4 presents our software and scientific methodology contributions, which aim to enhance the scientific rigor of MORL research and facilitate the application of these methods in real-world scenarios.

Part III: Multi-Agent Multi-Objective Reinforcement Learning

Part III extends our previous work to the domain of multi-objective multi-agent reinforcement learning. First, Chapter 5 focuses on algorithmic contributions to demonstrate the transferability of MORL and MARL methods to the MOMARL domain. It also provides software tools to establish a robust foundation for this emerging research field. Then, Chapter 6 applies these methods to real-world scenarios.

Part IV: Conclusion

Finally, Part IV concludes this thesis by going over the identified research questions in light of the presented contributions. Future research directions identified throughout this PhD work are also discussed.

Part I

Background & State of the Art

Chapter 2

Background and State of the Art

Contents

This chapter introduces notations and state of the art required for the remainder of the thesis. Each ensuing section provides formal definitions, notations, solution concepts, and significant insights pertinent to the respective fields addressed in this study. Section 2.1 focuses on the domain of reinforcement learning, while Section 2.2 delves into multi-objective optimization. Expanding upon these foundational domains, Section 2.3 introduces the field of multi-objective reinforcement learning. Finally, Section 2.4 adds a final dimension of complexity with multi-objective multi-agent reinforcement learning.

2.1 Reinforcement Learning

In the framework of RL, an agent interacts with an environment as follows: when faced with a particular state, the agent selects an action, which, upon execution, alters the environment, and in return, the agent receives a reward that indicates the quality of the action taken (see Figure 2.1). In this context, the primary objective of the agent is to acquire a policy function, typically denoted by π, that optimally guides its action choices to maximize the cumulative rewards obtained during its interactions with the environment. After the training period, this learned policy dictates the agent's decision-making process when it encounters different states [Sutton and Barto, 2018].

2.1.1 Formal Definitions and Notations

Formally, an RL problem is modeled by a Markov decision process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, r, p, \gamma, \mu_0)$ where:

- $\mathcal{S}$ are the states the agent can perceive;
- $\mathcal{A}$ are the actions the agent can undertake to interact with the environment;
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function;
- $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the probability of transition function, giving the probability of the next state given the current state and action;
- $\gamma \in [0, 1]$ is the discount factor, specifying the relative importance of long-term rewards with respect to short-term rewards;
- μ_0 is the distribution over initial states s_0.

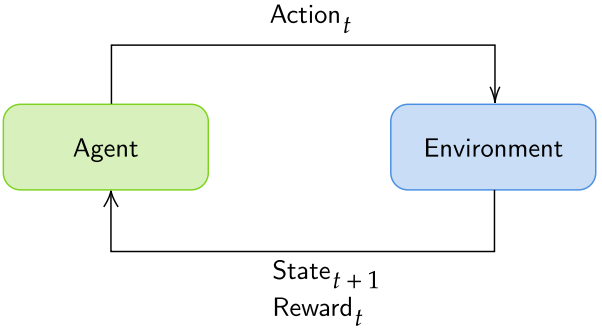


FIGURE 2.1: Reinforcement learning agent-environment interactions.

2.1.2 Solution Concepts

Within this framework, a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ can be assigned a numerical value upon evaluation in the MDP [Sutton and Barto, 2018]. This evaluation value is formally referred to as the value function and can be expressed as the discounted sum of rewards collected over an infinite time horizon (or until the end of an episode) starting from a given state s:

$$v^{\pi}(s) \equiv \mathbb{E}_{a_t \sim \pi(s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) |\ s_t = s \right], \tag{2.1}$$

where t represents the timesteps at which the agent makes a choice. Furthermore, the overall value of a policy π, regardless of the initial state, can be expressed as:

$$v^{\pi} \equiv \mathbb{E}_{s_0 \sim \mu_0} v^{\pi}(s_0). \tag{2.2}$$

Such a value is traditionally used to define an ordering on policies, *i.e.*, $\pi \succeq \pi' \iff v^{\pi} \geq v^{\pi'}$. The goal of an RL algorithm is then to find an optimal policy π^*, defined as one which maximizes v^{π}: $\pi^* = \arg\max_{\pi} v^{\pi}$. Similarly, the value of taking a particular action a in a given state s is expressed as:

$$q^{\pi}(s, a) \equiv \mathbb{E}_{a_t \sim \pi(s_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) |\ s_t = s, a_t = a \right]. \tag{2.3}$$

And there is a direct relation between v^{π} and q^{π}:

$$v^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\, q^{\pi}(s, a) \tag{2.4}$$

Algorithm 2.1 Reinforcement Learning process.

Input: Stopping criterion *stop*, Environment *env*.
Output: An optimized policy π.

1: $\pi = Initialize()$
2: $\mathcal{B} = \emptyset$
3: **while** $\neg stop$ **do**
4: $\quad s = env.reset(), done = False$
5: $\quad$ **while** $\neg done$ **do**
6: $\quad\quad a = Sample(\pi, s)$
7: $\quad\quad s', r, done = env.step(a)$
8: $\quad\quad \mathcal{B} = UpdateBuffer(\mathcal{B}, (s, a, r, s'))$
9: $\quad\quad$ **if** *update* **then**
10: $\quad\quad\quad \tilde{v}^{\pi} = EvaluatePolicy(\pi)$
11: $\quad\quad\quad \pi = ImprovePolicy(\pi, \tilde{v}^{\pi}, \mathcal{B})$
12: $\quad\quad$ **end if**
13: $\quad\quad s = s'$
14: $\quad$ **end while**
15: **end while**
16: **return** π

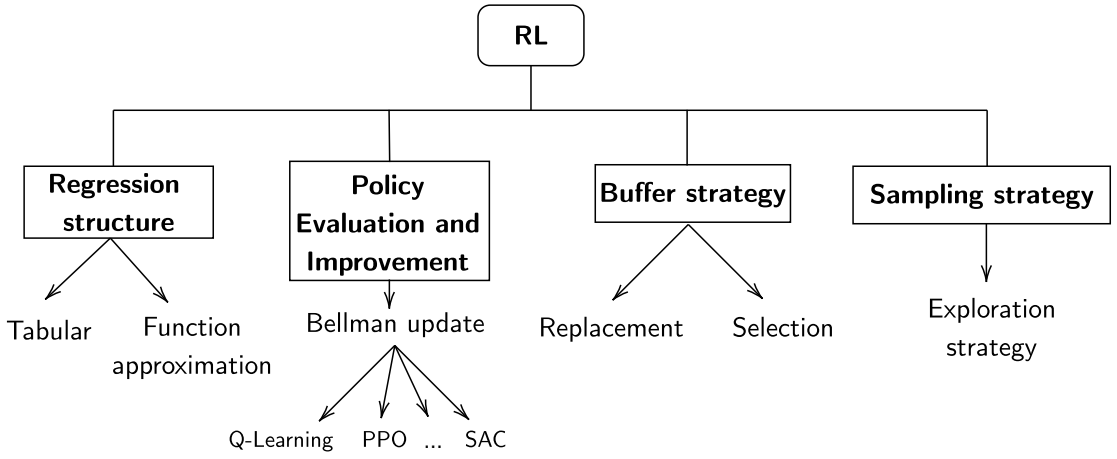


FIGURE 2.2: Reinforcement Learning: design choices

2.1.3 Solving Methods

To find such an optimal policy, many RL algorithms have been published over the last decades. A high-level skeleton of the RL process is presented in Algorithm 2.1. The algorithm first initializes its policy and an experience buffer (lines 1–2). Then, it samples experience tuples (s_t, a_t, r_t, s_{t+1}) from the environment by using the current policy and stores those in an experience buffer (lines 6–8). From the buffered experiences and its current estimated value, the policy is improved (lines 10–11). The optimization process stops when a criterion specified by the user is met and the current policy is returned.

Each part of the algorithm can be instantiated in various ways, constituting the design choices of RL according to our proposed taxonomy illustrated in Figure 2.2. The following sections discuss the role of each part and give a few examples of possible instantiations.

2.1.3.1 Regression Structure

A cornerstone design point in RL regards how to encode the policy function. Early RL algorithms often represented the policy using a tabular format. For example, Q-Learning [Watkins and Dayan, 1992] stores estimations of the action-values, $\tilde{q}(s, a)$ in such a table (see Equation 2.5 below for a formal definition). From this structure, a policy can be derived by selecting the action with the highest Q-value from each state, *i.e.*, $\pi(s) = \arg\max_{a \in \mathcal{A}} \tilde{q}(s, a)$. This is known as a greedy deterministic policy, as it always chooses the action with the largest expected reward.

However, this kind of approach does not scale to highly dimensional problems, *e.g.*, with continuous states or actions. Hence, recent algorithms use function approximations, such as regression trees or deep neural networks (DNNs), *e.g.*, deep Q-Network (DQN) [Mnih et al., 2015]. When employing DNNs, this method is commonly referred to as deep RL. In scenarios where the policy, denoted as π_θ, is parameterized with θ representing a set of parameters and Θ being the parameter space, the RL problem transforms into the task of identifying the optimal assignment of parameters. Mathematically, this is expressed as $\theta^* = \arg\max_{\theta \in \Theta} v^{\pi_\theta}$.

Finally, recent algorithms often leverage multiple regression structures for enhanced performance. For instance, in actor-critic settings, the actor structure aims to represent the probability of taking an action (*i.e.*, the policy), while another structure, the critic, estimates state(-action) values and facilitates bootstrapping [Sutton and Barto, 2018].

2.1.3.2 Policy Evaluation and Improvement

RL algorithms usually rely on estimated values to bootstrap their policy improvement process [Sutton and Barto, 2018]. For example, from an experience tuple (s_t, a_t, r_t, s_{t+1}) sampled from the environment and its current estimations $\tilde{q}(s, a)$ of the Q-value (see Equation 2.3), the well-known Q-Learning algorithm [Watkins and Dayan, 1992] updates its estimations using the following relation:

$$\tilde{q}(s_t, a_t) \leftarrow \tilde{q}(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a' \in \mathcal{A}} \tilde{q}(s_{t+1}, a') - \tilde{q}(s_t, a_t) \right) \tag{2.5}$$

where $\alpha \in [0, 1]$ is the learning rate, and the term $r_t + \gamma \max_{a' \in \mathcal{A}} \tilde{q}(s_{t+1}, a') - \tilde{q}(s_t, a_t)$ is called temporal difference error (TD-error). This TD-error typically measures the error between the estimated action-values and actual action-values sampled from the environment. By repeatedly sampling experiences from the environment and updating its policy, the agent is able to converge to an optimal policy in an infinite horizon (in tabular settings) [Sutton and Barto, 2018]. Many other update relations have been published over the years, allowing to learn more efficiently in challenging domains: policy gradient methods such as REINFORCE [Sutton and Barto, 2018], or actor-critic approaches such as proximal policy optimization (PPO) [Schulman et al., 2017] and soft actor-critic (SAC) [Haarnoja et al., 2018].

2.1.3.3 Buffer Strategy

In recent algorithms, policy updates are often batched using an experience buffer. This allows learning multiple times from an experience by replaying it [Lin, 1992], but also speeding up the policy improvement steps by performing mini-batch gradient descent in deep RL [Goodfellow et al., 2016]. Two choices linked to buffers arise: deciding which experiences in the buffer will be replaced by new ones, and which experiences to select to update the policy.

Replacement. While the simplest replacement criterion is based on recency, more elaborate techniques have also been studied. For example, de Bruin et al. [2016] propose using two experience buffers: one that is close to the current policy (recency criterion), while another one keeps the stored experiences close to a uniform distribution over the state-action space. This allows the computation of more robust policies and reduces the need for continuous, thorough exploration.

Selection. The most straightforward method for selecting experiences from the buffer is to choose them uniformly. However, research has shown in various articles that more intelligent experience selection strategies can significantly improve results in practice. For instance, prioritized sampling, as discussed in Schaul et al. [2016], is one such approach that has been shown to yield superior outcomes.

2.1.3.4 Sampling Strategy

The performance of an RL agent depends on which experiences were collected in the environment to learn its policy. Thus, the question of which action to choose in each state is crucial in such algorithms. To reach good performances, an agent must ideally sample (1) multiple times the same state-action pairs so as to compute good estimates of the environment dynamics (which can be stochastic), (2) regions leading to good rewards, to obtain good performances, and (3) unexplored regions, to find potentially better regions. The tension between points (2) and (3) is commonly referred to as the *exploration-exploitation dilemma*. Typically, the agent faces the dilemma of strictly adhering to its current policy (exploitation) and making exploratory moves guided by specific criteria.

Due to the critical impacts of such decisions on RL performance, many additional techniques have been proposed to tackle this issue [Amin et al., 2021, Zhang et al., 2023]. Various methods have been proposed to make the agent exploit the already gained knowledge while remaining curious about new areas. The most common exploration strategies are ϵ-greedy, Boltzmann (softmax), optimistic initialization, upper confidence bounds, and Thompson sampling [Vamplew et al., 2017a, Sutton and Barto, 2018]. Additionally, certain methods suggest building a model, such as a map, of the environment. This model provides the agent with a broader perspective, enabling more systematic exploration strategies. This concept is referred to as "state-based exploration" in the work of Moerland et al. [2023].

2.1.4 Use Cases Scenarios

Recently, (deep) RL has found application in various problem domains, particularly those involving sequential decision-making. Traditional areas where this technique is commonly employed include playing board games [Silver et al., 2016] and video games [Mnih et al., 2015, Wurman et al., 2022]. Beyond these domains, RL has been extended to real-world situations that demand rapid real-time responses (akin to Kahneman's System 1 [Kahneman, 2012]), such as drone racing [Kaufmann et al., 2023], large language models [Ouyang et al., 2022], or the control of tokamak plasmas in nuclear fusion reactors [Degrave et al., 2022, Tracey et al., 2024].

2.1.5 Summary

This section presented RL, which automates the learning process of agents' behaviors. The whole learning process is based on reinforcement towards good signals, provided by the reward function. In particular, it introduced a novel taxonomy (see Figure 2.2) for discerning design choices within RL algorithms and discussed

notable contributions corresponding to each design choice. Conveniently, the taxonomy allows deriving various RL algorithms by making atomic changes within its categories. Yet, in MDPs, the reward functions are limited to scalar signals, which limits the applicability of such techniques. Indeed, real-world scenarios often involve making compromises between multiple objectives, as noted in Vamplew et al. [2022]. To initiate the discussion towards more complex reward schemes, the following section presents the domain of multi-objective optimization.

2.2 Multi-Objective Optimization

Multi-objective optimization (MOO) addresses challenges involving multiple conflicting objectives, representing solutions through decision variables assigned to specific values denoted as $\mathbf{x}$. The evaluation of these solutions relies on a vector function $\mathbf{f}$ with m dimensions, where m corresponds to the number of objectives being considered. In contrast to RL, which typically tackles dynamic problems, multi-objective optimization is centered on static problems.[1] In static problems, the solution involves assigning decision variables specific values to optimally meet the objectives, rather than constituting a reusable function.

2.2.1 Formal Definitions and Notations

Formally, a multi-objective optimization problem (MOP) is expressed as:

$$\max \mathbf{f}(\mathbf{x}) = \max_{\mathbf{x}\in\Omega}(f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})). \tag{2.6}$$

Here, Ω represents the decision space (also known as the search space), and $\mathbf{f} : \Omega \mapsto \mathbb{R}^m$ denotes the objective function, comprising the individual objective functions $f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})$. It is worth noting that problems involving minimization can be straightforwardly converted into maximizing problems by maximizing the negated objective function.

2.2.2 Solution Concepts

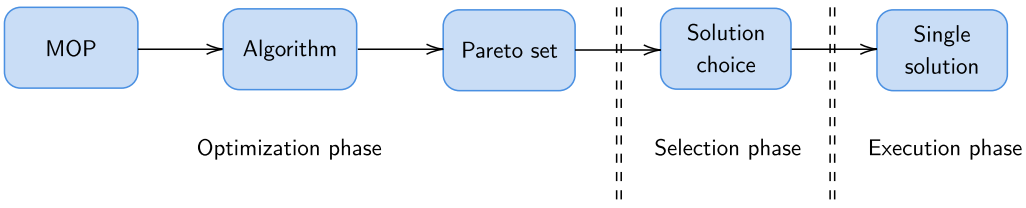


FIGURE 2.3: Different phases of the decision-making process in the *a posteriori* setting. Additional information can be found in Hayes et al. [2022].

In cases where the decision maker (DM) has known preferences over objectives *a priori*, these problems can be simplified to single-objective problems by converting the objective values into scalars using a scalarization function $g : \mathbb{R}^m \mapsto \mathbb{R}$. However, many approaches and real-life scenarios cannot make this assumption and instead operate within the *a posteriori* setting, where the DM's preferences are not known in advance.

In the *a posteriori* setting, where the evaluation of each solution maintains a vector shape, algorithms commonly rely on the concept of Pareto dominance to establish an order among solutions. An evaluation vector $\mathbf{f}(\mathbf{x})$ is

[1] While there are MOO algorithms capable of addressing dynamic problems, they are not the primary focus of this field of study.

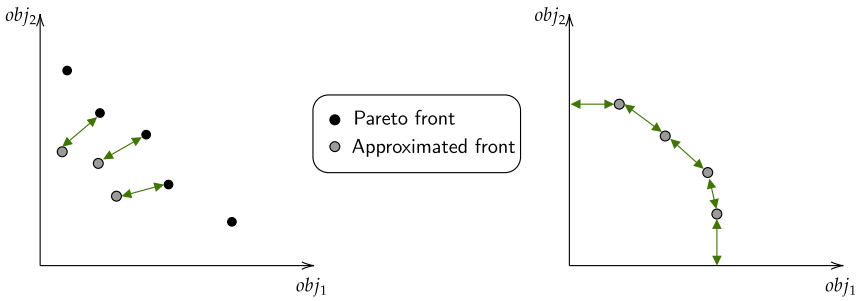


FIGURE 2.4: Illustration of Pareto front approximations. In the left part, the convergence aspect of the approximated front is represented by the arrows. In the right part, the diversity aspect is represented by the arrows.

said to Pareto dominate another vector $\mathbf{f}(\mathbf{x}')$ if and only if it is strictly better for at least one objective, without being worse in any other objective. Formally:

$$\mathbf{f}(\mathbf{x}) \succ_P \mathbf{f}(\mathbf{x}') \iff (\forall i : f_i(\mathbf{x}) \geq f_i(\mathbf{x}')) \wedge (\exists j : f_j(\mathbf{x}) > f_j(\mathbf{x}')). \tag{2.7}$$

This condition can also be relaxed to what is called *weak dominance*, denoted $\mathbf{f}(\mathbf{x}) \succeq_P \mathbf{f}(\mathbf{x}')$, whenever

$$\mathbf{f}(\mathbf{x}) \succeq_P \mathbf{f}(\mathbf{x}') \iff \forall i : f_i(\mathbf{x}) \geq f_i(\mathbf{x}'). \tag{2.8}$$

Pareto dominance does not impose a total ordering on all solutions. For instance, it is possible that two solutions' evaluations, such as $(1, 0)$ and $(0, 1)$, may not dominate each other. These solutions are referred to as *Pareto optimal*, signifying that they are both potentially optimal solutions as long as the DM's preferences remain unknown. Consequently, the primary goal of the optimization process is to identify a collection of Pareto optimal solutions, referred to as the *Pareto set* (PS). The evaluations associated with these solutions constitute the *Pareto front* (PF). Upon receiving a set of solutions, the DM can then make an informed choice, considering the trade-offs presented in the PF (see Figure 2.3). Formally, a PS is defined as follows:

$$\mathcal{PS} \equiv \{\mathbf{x} \mid \nexists\, \mathbf{x}' \text{ s.t. } \mathbf{f}(\mathbf{x}') \succ_P \mathbf{f}(\mathbf{x})\}. \tag{2.9}$$

Similarly, its associated PF, $\mathcal{F}(\mathcal{PS})$[2], is defined as follows:

$$\mathcal{F}(\mathcal{PS}) \equiv \{\mathbf{f}(\mathbf{x}) : \forall\, \mathbf{x} \in \mathcal{PS}\}. \tag{2.10}$$

Approximated optimization methods. In most scenarios, real-world problems are often too hard to solve using exact methods. Hence, algorithms often provide an approximation of the Pareto set (and its corresponding front) by relying on metaheuristics [Talbi, 2009]. A good approximation of the PF is characterized by two criteria: (1) *convergence* with the true solution, to present solutions of good quality to the DM, and (2) *diversity*, to present a wide range of compromises to the DM. Examples of Pareto fronts showing the importance of both criteria are shown in Figure 2.4.

[2]For concision, we often omit to specify the parameter $\mathcal{PS}$ since it is implicitly linked. In such cases, we denote the PF as $\mathcal{F}$.

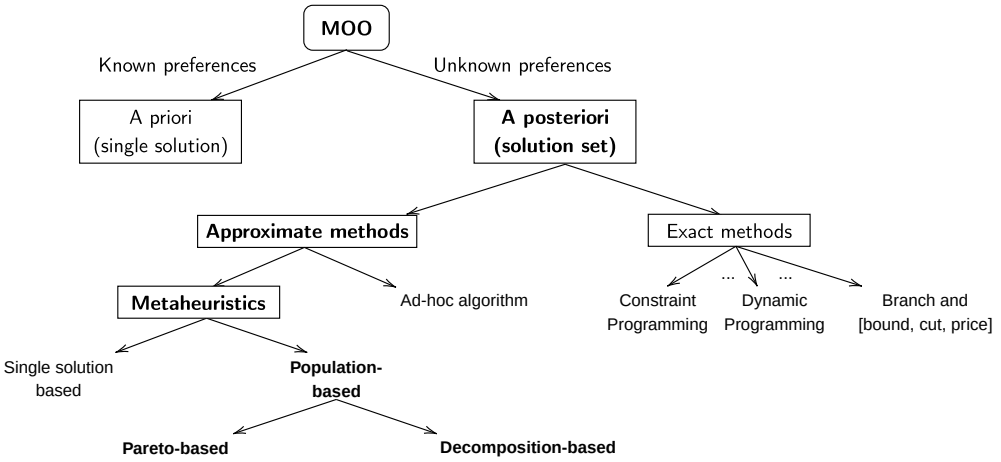


FIGURE 2.5: A non-exhaustive mapping of multi-objective optimization methods, our focus in this thesis is highlighted in bold fonts. Adapted from Talbi [2009].

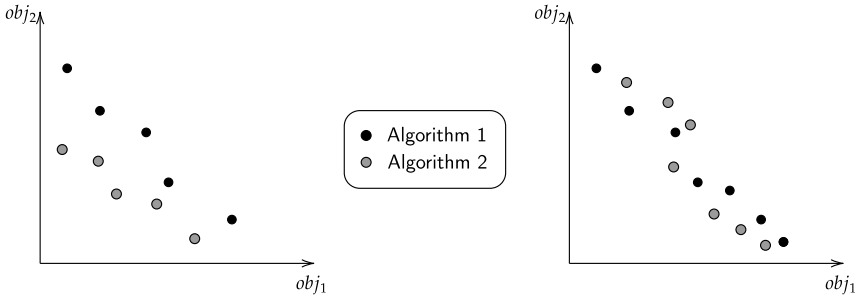


FIGURE 2.6: Comparison between PF approximations obtained by two different algorithms. Left: the first algorithm is clearly better than the second, right: there is no best algorithm since the PFs are intertwined.

Single-solution and population-based methods. The literature of *a posteriori* MOO is generally divided into two classes of algorithms: *single solution based* and *population-based* [Talbi, 2009]. The first class maintains and improves a single solution at a time and loops through various preferences, trajectories, or constraints to discover multiple solutions on the PF, *e.g.*, Czyżak and Jaszkiewicz [1998], Hansen [2000]. The second class maintains a set of solutions, called *population*, that are jointly improved over the search process, *e.g.*, Zhang and Li [2007], Alaya et al. [2007]. Notably, this population-based approach is currently considered the state of the art for solving MOPs due to the performance of these algorithms. Consequently, this work will concentrate on population-based algorithms. Sections 2.2.4 and 2.2.5 discuss these population-based solving methods in more detail. Finally, Figure 2.5 visually highlights the focus of this thesis on the global landscape of MOO solving methods. A thorough discussion on MOO solving techniques can be found in Talbi [2009].

2.2.3 Performance Indicators

The assessment and comparison of PFs obtained by different algorithms pose challenges due to the PFs being a collection of points. This task is not straightforward for two main reasons. Firstly, the presented PFs can be intertwined, as illustrated in Figure 2.6. Secondly, PFs in high dimensions present difficulty in visualization. In practice, *performance indicators* become essential to transform a PF into a scalar value. This enables comparisons and establishes an order among PFs. Various types of performance indicators are available for this

purpose. However, it is important to note that compressing a set of points into a single scalar value cannot be achieved without bias. Hence, multiple performance indicators (assessing different criteria) are often used in practice when comparing PFs.

Performance indicators can be categorized into two groups: *axiomatic indicators*, which do not make any assumption about the DM's utility, and *utility-based indicators*, which presuppose a specific shape for the DM's utility. Several of these indicators, employed throughout this dissertation, are illustrated below. Note that ↑ means the higher the indicator value is, the better the corresponding PF is, whereas ↓ is the opposite.

2.2.3.1 Axiomatic Indicators

These indicators do not make any assumption on the shape of the DM's utility.

Inverted generational distance (↓, convergence). This metric quantifies the convergence rate of an approximate PF, $\tilde{\mathcal{F}}$, towards a reference (potentially optimal) PF, $\mathcal{Z}$ [Coello Coello and Reyes Sierra, 2004]. If the reference front is unknown, it is usually defined or constructed by aggregating the best value vectors observed after several executions of the studied optimization algorithm. The inverted generation distance (IGD) is computed as:

$$\mathrm{IGD}(\tilde{\mathcal{F}}, \mathcal{Z}) = \frac{1}{|\mathcal{Z}|}\sqrt{\sum_{\mathbf{f}(\mathbf{x}^*)\in\mathcal{Z}} \min_{\mathbf{f}(\mathbf{x})\in\tilde{\mathcal{F}}} \|\mathbf{f}(\mathbf{x}^*) - \mathbf{f}(\mathbf{x})\|^2}.$$

Cardinality (↑, diversity). This metric is computed by considering the count of solutions within the approximated PF ($\tilde{\mathcal{F}}$). It offers insights into the diversity of $\tilde{\mathcal{F}}$ by indicating the number of trade-offs identified by the algorithm.

$$\mathrm{C}(\tilde{\mathcal{F}}) = |\tilde{\mathcal{F}}|.$$

Sparsity (↓, diversity). This metric also characterizes the diversity of the solutions within a given $\tilde{\mathcal{F}}$. It computes the distance between each consecutive point in $\tilde{\mathcal{F}}$ after sorting by objective [Xu et al., 2020a]. The sparsity of an approximate PF, $\tilde{\mathcal{F}}$, is given by:

$$\mathrm{S}(\tilde{\mathcal{F}}) = \frac{1}{|\tilde{\mathcal{F}}| - 1}\sum_{j=1}^{m}\sum_{i=1}^{|\tilde{\mathcal{F}}|-1} (\mathcal{L}_j(i) - \mathcal{L}_j(i+1))^2,$$

where $\mathcal{L}_j$ is the sorted list of the values of the j-th objective considering all solutions' evaluations in $\tilde{\mathcal{F}}$, and $\mathcal{L}_j(i)$ is the i-th value in $\mathcal{L}_j$.

Hypervolume (↑, hybrid). This is a hybrid metric quantifying both a PF's convergence and diversity. Given an approximate PF, $\tilde{\mathcal{F}}$, and a pessimistic reference point, $\mathbf{z}_{\mathrm{ref}}$, the hypervolume indicator represents the volume of the objective space portion starting from $\mathbf{z}_{\mathrm{ref}}$ that is weakly dominated by $\tilde{\mathcal{F}}$. Formally, the hypervolume metric [Zitzler, 1999] is defined as:

$$\mathrm{HV}(\tilde{\mathcal{F}}, \mathbf{z}_{\mathrm{ref}}) = \Lambda\left(\bigcup_{\substack{\mathbf{f}(\mathbf{x})\in\tilde{\mathcal{F}} \\ \mathbf{f}(\mathbf{x})\succeq_P \mathbf{z}_{\mathrm{ref}}}} Box(\mathbf{f}(\mathbf{x}), \mathbf{z}_{\mathrm{ref}})\right),$$

where $\Lambda(\cdot)$ is the Lebesgue measure and $Box(\mathbf{f}(\mathbf{x}), \mathbf{z}_{\mathrm{ref}}) = \{\mathbf{p} \in \mathbb{R}^m \mid \mathbf{f}(\mathbf{x}) \succeq_P \mathbf{p} \succeq_P \mathbf{z}_{\mathrm{ref}}\}$ denotes the box delimited above by $\mathbf{f}(\mathbf{x}) \in \tilde{\mathcal{F}}$ and below by $\mathbf{z}_{\mathrm{ref}}$. The reference point used in the hypervolume computation is typically an estimate of the worst-possible value per objective.

Algorithm 2.2 Pareto-based MOO

Input: Population size n, stopping criterion *stop*
Output: The approximate Pareto Set, $\mathcal{PS}$.

1: $\mathcal{P} = InitPopulation()$
2: $\mathcal{PS} = ParetoPrune(\mathcal{P}, \mathcal{F}(\mathcal{P}))$
3: **while** $\neg stop$ **do**
4: Choose candidates $\mathcal{C}$ from $\mathcal{P}$ based on fitness computed from $\mathcal{F}(\mathcal{P})$
5: $\mathcal{C} = Crossover(\mathcal{C})$
6: $\mathcal{C} = Mutation(\mathcal{C})$
7: $\mathcal{PS} = ParetoPrune(\mathcal{PS} \cup \mathcal{C}, \mathcal{F}(\mathcal{PS}) \cup \mathcal{F}(\mathcal{C}))$
8: Update population $\mathcal{P}$ by selecting from $\mathcal{C} \cup \mathcal{P}$ (limit size n)
9: **end while**
10: **return** $\mathcal{PS}$

2.2.3.2 Utility-Based Indicators

These indicators make assumptions on the shape of the DM's utility, *e.g.*, that the user has a linear utility function. Notably, the illustrated metrics are close to the R-metrics family in MOO [Talbi, 2009].

Expected utility (↑). In the case where the utility function of the DM, u, is linear, it becomes feasible to represent the expected utility over a distribution of reward weights, $\mathcal{W}$, using the expected utility (EU) metric [Zintgraf et al., 2015]. The EU metric is then defined as:

$$\mathrm{EU}(\tilde{\mathcal{F}}) = \mathbb{E}_{\mathbf{w}\sim\mathcal{W}}\left[\max_{\mathbf{f}(\mathbf{x})\in\tilde{\mathcal{F}}} \mathbf{f}(\mathbf{x})\cdot\mathbf{w}\right].$$

Maximum utility loss (↓). The metric introduced by Zintgraf et al. [2015] serves to quantify the maximum utility loss (MUL) arising from the use of an approximate PF ($\tilde{\mathcal{F}}$) instead of a reference PF ($\mathcal{Z}$). It is defined as follows:

$$\mathrm{MUL}(\tilde{\mathcal{F}}, \mathcal{Z}) = \max_{\mathbf{w}\in\mathcal{W}}\left(\max_{\mathbf{f}(\mathbf{x}^*)\in\mathcal{Z}} \mathbf{f}(\mathbf{x}^*)\cdot\mathbf{w} - \max_{\mathbf{f}(\mathbf{x})\in\tilde{\mathcal{F}}} \mathbf{f}(\mathbf{x})\cdot\mathbf{w}\right).$$

2.2.4 Pareto-Based Methods

The first approach to solving an MOP, called *Pareto-based* (see Figure 2.5), typically involves the implementation of population-based algorithms, such as genetic algorithms (GA). These can generate a diverse set of Pareto optimal solutions in a single run. Moreover, contrary to other methods such as decomposition-based (discussed in Section 2.2.5), Pareto-based methods are not sensitive to the shape of PF (continuity, convexity, etc.). They generally leverage the concept of Pareto dominance (Equation 2.7) to keep a Pareto optimal population of solutions.

Algorithm 2.2 gives a high-level skeleton of such Pareto-based techniques for MOO. The algorithm starts by initializing a population of solutions and uses the *ParetoPrune* function (see Equation 2.9) to remove all dominated solutions and form an external archive population (lines 1–2). Then at each generation, a set of candidates is chosen based on some fitness (discussed below), and crossover and mutation operators from the GA process are applied (lines 4–6). After these steps, the new candidates are evaluated and added to the archive if they contribute to an improved PF (line 7). The population is updated based on these new candidates and the old population (line 8). Finally, the elite population, forming the PS, is returned.

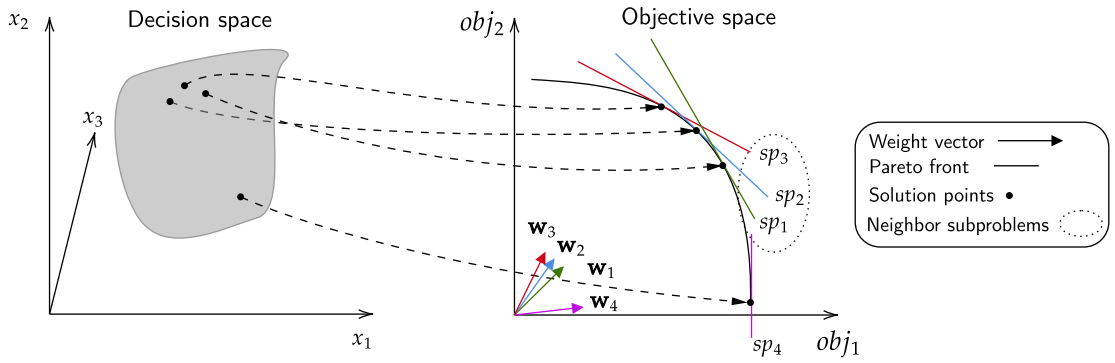


FIGURE 2.7: The decomposition in the objective space idea: split the multi-objective problem into various single-objective problems sp_n by relying on a scalarization function (weighted sum in this case). sp_1, sp_2, and sp_3 are considered to be neighbors since their associated weight vectors are close to each other while sp_4 is not considered to be in the neighborhood.

Because the presented algorithm is flexible, some parts have potentially different implementations. For instance, there exist many different crossover and mutation operators in the GA literature [Talbi, 2009]. It is interesting to note that these operators provide different levels of exploration-exploitation, similar to RL strategies (Section 2.1.3.4). The following sections describe design choices that are related to the MO parts, *i.e.*, fitness metric, and Pareto archive.

2.2.4.1 Fitness Metrics

Within the population of solutions, a *fitness metric* is employed to quantify the contribution of a given solution. Typically, solutions deemed important (with high fitness) are selected for *crossover* and *mutations* in the subsequent iteration. These fitness metrics can be computed in two ways: (1) **based on Pareto dominance** concepts, as observed in algorithms like the non-dominated sorting genetic algorithm (NSGA-II) [Deb et al., 2002] (*e.g.*, assessing how many solutions in the population the current solution dominates); or (2) **based on performance indicators**, exemplified by the indicator-based evolutionary algorithm (IBEA) [Zitzler and Künzli, 2004] (*e.g.*, measuring the change in a given performance indicator if the current solution is removed from the Pareto set).

2.2.4.2 Archive

At a given iteration, the algorithm may find a worse solution after applying its search step (crossover and mutation). This could lead to a degradation of the population's quality. To solve this issue, modern algorithms often rely on the concept of external archive population ($\mathcal{PS}$) which stores the non-dominated solutions seen so far. The Pareto dominance criterion (Equation 2.9) can be used as a pruning function (*ParetoPrune*) to determine which individuals to keep in the archive. In case there are too many incomparable solutions, the size of the archive can be limited by using additional techniques, *e.g.*, crowding distance [Deb et al., 2002].

2.2.5 Decomposition-Based Methods

Another way to solve MOPs is to decompose the MOP into several single-objective problems (SOPs) by using a scalarization function. This function, represented by $g : \mathbb{R}^m \times \mathbb{R}^m \mapsto \mathbb{R}$, employs associated weights represented

Algorithm 2.3 Multi-objective optimization based on Decomposition (MOO/D).

Input: Stopping criterion *stop*, Scalarization method g, Exchange trigger *exch*.
Output: The approximation of the Pareto set stored in the external archive population $\mathcal{PS}$.

1: $\mathcal{P}, \mathcal{W}, \mathcal{Z} = Initialize()$
2: $\mathcal{PS} = ParetoPrune(\mathcal{P}, \mathcal{F}(\mathcal{P}))$
3: $\mathcal{N} = InitializeNeighborhood(\mathcal{P}, \mathcal{W})$
4: **while** $\neg stop$ **do**
5: $\quad \mathcal{C}, \mathcal{W}', \mathcal{Z}' = Select(\mathcal{P}, \mathcal{W}, \mathcal{Z})$
6: $\quad \mathcal{C} = Search(\mathcal{C}, \mathcal{W}', \mathcal{Z}', g, exch)$
7: $\quad \mathcal{P} = \mathcal{P} \cup \mathcal{C}$
8: $\quad \mathcal{PS} = ParetoPrune(\mathcal{PS} \cup \mathcal{C}, \mathcal{F}(\mathcal{PS}) \cup \mathcal{F}(\mathcal{C}))$
9: $\quad \mathcal{W}, \mathcal{Z} = Adapt(\mathcal{W}, \mathcal{Z}, \mathcal{F}(\mathcal{PS}))$
10: $\quad \mathcal{N} = UpdateNeighborhood(\mathcal{P}, \mathcal{N}, \mathcal{W})$
11: $\quad Cooperate(\mathcal{N})$
12: **end while**
13: **return** $\mathcal{PS}$

by the vector $\mathbf{w}$. This methodology, as depicted in Figure 2.7, facilitates the approximation of the PF by solving the SOPs corresponding to different weight vectors. Each of these weight vectors is designed to target a specific region of the PF, providing a comprehensive exploration of the objective space. Moreover, decomposition usually simplifies the problem and offers a simple way to parallelize the search process. It is important to note that the generated SOPs may then be solved by single-objective optimization solvers, allowing to easily transfer improvements from the field of single-objective optimization to the field of MOO. This technique is applicable in both the context of single-solution-based and population-based algorithms. In the population-based setting, the assumption that neighboring subproblems share common solution components is often utilized, enabling these subproblems to cooperate by exchanging information. This cooperation typically improves the optimization process.

A generic framework relying on such decomposition techniques can be found in Algorithm 2.3. MOO/D maintains a population of solutions $\mathcal{P}$, a set of weights $\mathcal{W}$, and reference points $\mathcal{Z}$ to apply in the scalarization function g. Similar to Pareto-based methods, the best individuals are kept in an external archive population $\mathcal{PS}$ during the optimization process according to a criterion defined by the *ParetoPrune* function. At each iteration, a set of individuals from the population, along with weights ($\mathcal{W}' \subseteq \mathcal{W}$) and reference points ($\mathcal{Z}' \subseteq \mathcal{Z}$) are selected as starting points to search for better solutions until an exchange criterion (*exch*) is triggered (lines 5–6). Then, the generated candidates are integrated into the population and archive based on their objective values (lines 7–8). Next, the algorithm adapts the weights and reference points according to the current state of the PF (lines 9–10). Additionally, some knowledge is exchanged between subproblems in the same neighborhood (line 11). Finally, the algorithm returns the set of non-dominated solutions seen so far (line 13).

Due to its successful application in numerous domains, many variations of the MOO/D framework have been proposed over the years. From the analysis of these variations, recurring patterns were identified, *e.g.*, Figure 2.8 has been compiled from various sources [Talbi, 2009, Santiago et al., 2014, Xu et al., 2020b]. The rest of this section explains each of the building blocks that can be instantiated in different ways in such a framework. Some illustrative examples and notable articles are also pointed out in the discussion. For further references, some surveys dedicated to decomposition techniques in MOO can be found in [Santiago et al., 2014, Xu et al., 2020b].

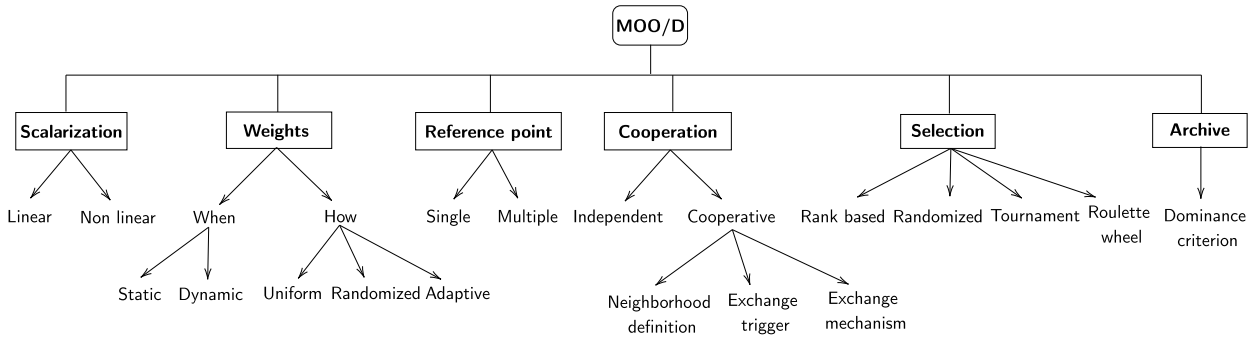


FIGURE 2.8: Design choices of multi-objective optimization based on decomposition (MOO/D).

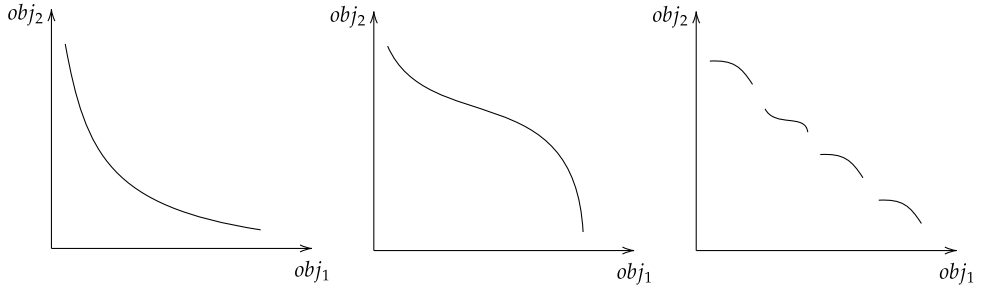


FIGURE 2.9: Examples of ill-shaped Pareto Fronts. Left: concave PF, middle: PF exposing convex and concave parts, right: discontinuous PF.

2.2.5.1 Scalarization Function

Various scalarization functions g have been the subject of studies over the last decades. These allow the MOP to be decomposed into multiple simpler SOPs. Moreover, scalarization functions are usually parameterized by weight vectors that target different areas of the objective space.

Linear scalarization. The most common and straightforward scalarization technique is the weighted sum: $g^{\text{ws}}(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{m} w_i f_i(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{f}(\mathbf{x})$. This method is easy to comprehend and enables the specification of weights as percentages to express preferences between the objectives. However, this approach has limitations. With this kind of simplification, the subproblems become linear, which means they cannot accurately capture points in the concave region of the PF [Marler and Arora, 2010] (see also Figure 2.9).

Non-linear scalarization. In response to the limitations of linear scalarization, alternative non-linear techniques, such as the Chebyshev scalarization, have been investigated. This scalarization function, also known as the norm L^∞, is defined as the maximum weighted distance to a utopian reference point $\mathbf{z}$, expressed as $g^{\text{ch}}(\mathbf{x}, \mathbf{w}, \mathbf{z}) = \max_{i \in [1,m]} |w_i(f_i(\mathbf{x}) - z_i)|$. Note that in this particular case, the goal is to minimize the scalarized values instead of maximizing like in the linear case. Using such non-linear techniques enables the identification of points within the concave regions of the PF. See for example the work of [Emmerich and Deutz, 2018] for some visual examples. Intriguingly, some studies, such as Ishibuchi et al. [2010], have also proposed the combination or adaptation of various scalarization techniques to leverage their respective advantages.

2.2.5.2 Weight Vectors

Scalarization functions rely on weight vectors $\mathbf{w} \in \mathcal{W}$ to target different areas in the objective space. In fact, the manner in which these weights are generated is crucial for obtaining quality solutions within a reasonable timeframe. For instance, when creating weights that target empty regions in discontinuous PFs, the optimization process inefficiently utilizes its computing resources. This inefficiency arises because the SOPs associated with these weights ultimately converge to the corners of the PF fragments (see Figure 2.9, right) resulting in duplicated computation to reach similar solution quality. Consequently, many ways to generate or adapt these weight vectors have been published. There are two design choices for the weights: (1) when and (2) how to (re-)generate them.

When to generate weights? The simplest approach is to fix the weights before any search is started (**static**). However, the shape of the Pareto front, being unknown at that time, can be complex and might require adapting the weights during the search to focus on sparse areas, *i.e.*, where the Pareto front is not well estimated. Thus, multiple ways to adapt the weights during the search have been published, *e.g.*, Qi et al. [2013]. This is referred to as **dynamic** weights in this work.

How to generate weights? Weights can be assigned through different approaches, such as **uniform** distribution across the objective space [Das and Dennis, 2000, Blank et al., 2021], **randomized** processes, or **adaptation** based on evolving knowledge during the search [Czyżżak and Jaszkiewicz, 1998, Qi et al., 2013]. Weight adaptation strategies can vary, including focusing on underrepresented regions in the estimated PF or predicting potential improvements in the objective space [Dubois-Lacoste et al., 2011]. Pareto simulated annealing (PSA) [Czyżżak and Jaszkiewicz, 1998], for instance, exemplifies this adaptation approach, as it adjusts an individual's weights in response to its current evaluation and proximity to non-dominated solutions. Formally, for an individual $\mathbf{x}$ and its nearest non-dominated neighbor $\mathbf{x}'$, PSA modifies the weights attached to the SOP leading to $\mathbf{x}$ using:

$$w_j^{\mathbf{x}} = \begin{cases} \delta w_j^{\mathbf{x}} & \text{if } f_j(\mathbf{x}) \geq f_j(\mathbf{x}') \\ w_j^{\mathbf{x}}/\delta & \text{if } f_j(\mathbf{x}) < f_j(\mathbf{x}'), \end{cases} \tag{2.11}$$

with δ being a constant close to 1, typically $\delta = 1.05$. This mechanism, encapsulated within the *Adapt* function of MOO/D, empowers the fine-tuning of SOPs to excel at objectives where they already exhibit proficiency. This, in turn, encourages the SOPs to move away from their neighboring solutions, ultimately enhancing the diversity within the estimated PF.

2.2.5.3 Reference Points

Certain scalarization techniques, such as Chebyshev, depend on the selection of an optimistic or pessimistic point in the objective space to serve as a reference point $\mathbf{z}$. Similarly to weight vectors, the choice of these reference points has a significant influence on the final performance of the MOO algorithm. The careful determination of these reference points is crucial, as it profoundly impacts the algorithm's ability to approximate the Pareto front effectively. Hence, multiple settings for reference points exist.

Single reference point. A straightforward approach to setting the reference point is to fix it before commencing the optimization process [Zhang and Li, 2007]. In this manner, the reference point remains constant throughout the optimization, allowing for a controlled and deterministic approach.

Multiple reference points. However, setting the reference point beforehand can sometimes result in suboptimal outcomes for the algorithm. For this reason, some studies suggest dynamic adaptation of the reference point during the search, as exemplified by the work of Liu et al. [2020]. This adaptive approach enables the reference point to evolve and align with the changing characteristics of the Pareto front, potentially improving the quality of the results.

2.2.5.4 Cooperation

The most straightforward approach to solving a MOP using decomposition is to **independently** address all generated SOPs. This concept underpins the first single-solution decomposition-based algorithms, such as the one introduced by Czyżak and Jaszkiewicz [1998]. Subsequently, the idea of promoting **cooperation** among subproblems in close proximity (referred to as neighbors) emerged [Zhang and Li, 2007]. Numerous studies have indicated that cooperation among these subproblems can accelerate the search process and yield superior performance. This is due to the fact that in practice, solutions to subproblems often share similarities in at least some of their components. Within this cooperative framework, several design choices come into play: (1) the definition of neighborhood relationships, (2) the timing of information exchange, and (3) the nature of the knowledge to be shared.

Neighborhood definition. A neighborhood $\mathcal{N}$ can be constituted of zero (no cooperation), multiple, or all other subproblems. The most common way to define neighborhoods between SOPs is to rely on the Euclidean distance of their associated weight vectors; see Figure 2.7. Yet, other techniques have also been published such as Murata et al. [2001], which relies on the Manhattan distance between weight vectors instead.

Exchange trigger. Information exchange between SOPs can occur at various moments throughout the search process, with the timing often determined by an exchange trigger (*exch* in Algorithm 2.3). There exist three primary approaches to information exchange. **Periodic exchange** involves regular and predetermined information sharing, such as at every iteration. **Adaptive exchange**, on the other hand, responds dynamically to events occurring in the search process, such as when specific improvements are achieved. Finally, **continuous exchange** is usually achieved through a shared data structure that constantly disseminates information to guide the search process for all neighboring SOPs.

Exchange mechanism. The method of cooperation, denoted as *Cooperate* in MOO/D, among subproblems is closely associated with the specific search algorithm being employed. Presently, many decomposition techniques are integrated with genetic algorithms, which perform information exchange among SOPs by employing crossover operators on the solutions. This enables the sharing of solution components and relevant data. An alternative approach involves sharing part of the search memory, as seen in the case of ant colony algorithms, where elements like the pheromone matrix are shared among subproblems to guide the search process [Ke et al., 2013].

2.2.5.5 Selection

As the weight vectors and reference points attached to individuals within the population $\mathcal{P}$ can change dynamically, an exponential number of combinations arises for each optimization step. To address this, various selection mechanisms (denoted as *Select* in MOO/D) have been developed to choose a subset of individuals, along with their associated weights and reference points, for each search iteration.

For example, existing approaches **rank** solutions according to their scalarized values and select the best ones observed thus far, using a static weight vector and reference point [Zhang and Li, 2007]. In contrast, another method involves random generation of new weight vectors, followed by **tournament-style selection** to determine which solution will initiate the local search phase [Ishibuchi and Kaige, 2004]. Alternative techniques propose **random selection** or systematic iteration through the available choices, akin to a **roulette wheel** selection process [Talbi, 2009].

2.2.6 Use Cases Scenarios

Due to its prolonged existence, MOO has found applications across various domains throughout the years. Common applications encompass **network optimization**, which involves minimizing costs, maximizing service levels, and optimizing resource utilization. It has been employed in diverse fields such as supply chain management [Altiparmak et al., 2006], telecommunications [Almasoud and Kamal, 2015], and water management [Halhal et al., 1997]. In the realm of **engineering design**, MOO addresses factors like production cost, reliability, and safety. Examples include the optimization of gas turbine stator blades [Hasenjäger et al., 2005] and aircraft design [Bandte and Malinchik, 2004]. Finally, in the domain of **production scheduling**, MOO for example aims to minimize makespan and maximize planned tasks [Ojstersek et al., 2020]. Contrary to RL, domains where MOO applies are rather static and usually allow for longer processing time before taking a decision, akin to Kahneman's System 2 [Kahneman, 2012].

2.2.7 Summary

Section 2.1 presented the RL problem, as well as its building blocks, and concluded by discussing its limited expressivity for multi-objective problems. Section 2.2 presented an overview of MOO which aims at solving multi-objective problems. Notably, two distinct paradigms for MOPs were discussed: Pareto-based techniques, which maintain a population of Pareto optimal solutions throughout the search process, and Decomposition techniques, which employ a scalarization function to convert the MOP into several SOPs, utilizing single-objective solvers to search for solutions.

There are a few notable differences between MOO and RL: (1) in MOO, the dynamics of the environment ($\mathbf{f}$, and the MOP model) are known by the algorithm and are generally deterministic, whereas RL involves solving sequential problems with unknown and potentially stochastic dynamics, (2) in MOO, the solutions are assignments of the decision variables of the problem, whereas they are reusable functions in RL.[3] The upcoming section introduces the field of multi-objective reinforcement learning (MORL), which bridges the gap between the two previously discussed fields.

[3]Generating reusable functions to optimize problems is known as Hyper-Heuristics in optimization [Burke et al., 2013].

2.3 Multi-Objective Reinforcement Learning

Multi-objective RL extends traditional RL in making the agent receive a vectorial reward instead of a scalar (see Figure 2.10). This simple extension introduces challenges coming from the realm of MOO into RL, as the agent now has to maximize multiple, potentially conflicting objectives. This section presents these challenges and current solving approaches.

2.3.1 Formal Definitions and Notations

To extend RL to multi-objective problems, MORL models the problem as a multi-objective MDP (MOMDP). An MOMDP alters the MDP (see Section 2.1.1) by replacing the reward function with a vectorial reward signal [Roijers and Whiteson, 2017], *i.e.*,

$$\mathbf{r} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}^m.$$

2.3.2 Solution Concepts

In MORL, as in classical RL, the evaluation of policies is based on a sequence of decisions, and one usually runs a policy π for a finite amount of time to estimate its vector value $\mathbf{v}^\pi$. This value, which can be considered as the equivalent of $\mathbf{f}(\mathbf{x})$ for MORL, is the direct vectorial adaptation of v^π. Formally, an MORL policy has a vector value that is computed using the following:

$$\mathbf{v}^\pi = \mathbb{E}_{a_t \sim \pi(s_t)} \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(s_t, a_t, s_{t+1}) \mid \mu_0 \right]. \tag{2.12}$$

From this definition, one can define a Pareto dominance relation on MORL policies:

$$\pi \succ_P \pi' \iff (\forall i : v_i^\pi \geq v_i^{\pi'}) \wedge (\exists j : v_j^\pi > v_j^{\pi'}). \tag{2.13}$$

As mentioned earlier, one of the key differences between MOO and RL lies in the fact that RL aims at learning (optimizing) a policy, while MOO aims at optimizing a Pareto set of solutions. In the middle ground, MORL aims at learning a Pareto set of policies, with each policy potentially being parameterized by $\theta \in \Theta$ under function approximation. Formally, the PS and PF in MORL are defined as:

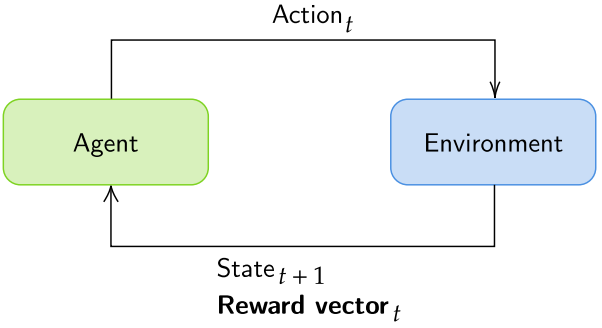


FIGURE 2.10: Multi-objective reinforcement learning agent-environment interactions.

Algorithm 2.4 Pareto Q-learning [Van Moffaert and Nowé, 2014]. Blue parts emphasize parts coming from RL.

Input: Stopping criterion *stop*, Environment *env*

Output: The QSets $\mathcal{Q}$.

$\mathcal{Q}(s,a) = \emptyset \; \forall s \in \mathcal{S}, a \in \mathcal{A}$
$n(s,a) = 0 \; \forall s \in \mathcal{S}, a \in \mathcal{A}$
$\overline{\mathbf{r}}(s,a) = \mathbf{0} \; \forall s \in \mathcal{S}, a \in \mathcal{A}$
while $\neg stop$ **do**
  $s = env.reset(), done = False$
  while $\neg done$ **do**
    Choose action a based on $\mathcal{Q}(s,a)$ ▷ Mix of *Sample* and fitness from Algorithms 2.1 and 2.2
    $s', \mathbf{r}, done = env.step(a)$
    $n(s,a) = n(s,a) + 1$
    $\mathcal{UF}(s') = ParetoPrune(\cup_{a' \in A} \mathcal{Q}(s', a'))$
    $\overline{\mathbf{r}}(s,a) = \overline{\mathbf{r}}(s,a) + \frac{\mathbf{r} - \overline{\mathbf{r}}(s,a)}{n(s,a)}$
    $\mathcal{Q}(s,a) = \overline{\mathbf{r}}(s,a) \oplus \gamma \mathcal{UF}(s')$ ▷ MO version of the Q-value update (Equation 2.5)
    $s = s'$
  end while
end while
return $\mathcal{Q}$

$$\mathcal{PS} \equiv \{\pi \mid \nexists \, \pi' \text{ s.t. } \pi' \succ_P \pi\}, \tag{2.14}$$

and

$$\mathcal{F}(\mathcal{PS}) \equiv \{\mathbf{v}^\pi : \forall \, \pi \in \mathcal{PS}\}. \tag{2.15}$$

2.3.3 Pareto-Based Methods

Similar to Pareto-based MOO (Section 2.2.4), certain MORL algorithms adopt an approach that involves directly learning a set of policies within the regression structure [Van Moffaert and Nowé, 2014, Ruiz-Montiel et al., 2017]. In these algorithms, the Q-values (see Equation 2.5) are adapted to store a set of Pareto optimal value vectors that can be achieved from the current state for a given action. The rest of this section illustrates such Pareto-based MORL algorithms with Pareto Q-Learning [Van Moffaert and Nowé, 2014]. Yet, it is worth noting that others exist, such as Ruiz-Montiel et al. [2017].

2.3.3.1 Example: Pareto Q-learning

Algorithm 2.4 depicts the learning process of Pareto Q-Learning [Van Moffaert and Nowé, 2014]. In this algorithm, the learning focus shifts from scalar values (Q-values, *e.g.*, Equation 2.5) to Pareto fronts (Q-sets) attainable for each state-action pair. Essentially, it determines which Pareto optimal points can be reached by taking a specific action in the current state and collecting the associated future rewards. The initialization

phase involves setting all Q-sets $\mathcal{Q}$ to empty sets, initializing a counter for the frequency of selecting a given state-action pair to 0, and initializing an average reward vector for each state-action pair (lines 1–3).

Then, the agent traverses the environment by selecting actions based on its current Q-sets (line 7). It is crucial to note that Q-sets are not scalar values, and extracting a policy using the arg max operator as in classical RL is not straightforward. In practice, the algorithm scalarizes these Q-sets by utilizing a performance indicator or dominance rank, akin to Pareto-based methods in MOO (refer to Section 2.2.4).

As the agent interacts with the environment, receiving the next state, a reward vector, and a completion signal, it updates its Q-sets. The process involves computing a unified PF for all potential actions in the next state s', similar to how v^π relates to q^π in Equation 2.4 (line 10), updating the average reward for the current state-action pair (line 11), and subsequently updating the corresponding Q-set by summing the average reward vector for the current state-action and the discounted PF of the next state (line 12). The operation $\oplus$ denotes a vector-sum between a vector and a set of vectors, adding the former to each vector in the set.

From the acquired Q-sets, a PF can be computed for a starting state s_0 by $\mathcal{F}(s_0) = ParetoPrune(\cup_{a' \in \mathcal{A}} \mathcal{Q}(s_0, a'))$ and exposed to the DM. Then, it becomes feasible to execute a policy leading to the chosen point by selecting the action whose Q-set contains the desired point. Subsequently, the received reward is subtracted from the desired point, guiding the selection of actions for subsequent states. It is worth noting that following learned policies in a stochastic environment poses an NP-Hard problem [Roijers et al., 2021]. Thus, this algorithm is usually employed in deterministic environments.

Pareto-based methods are very efficient where applicable. However, it is worth noting that these algorithms are currently limited to tabular structures and face challenges when applied to problems with high dimensions. Despite efforts to develop Pareto-based MORL algorithms using function approximation techniques, learning to produce sets of varying sizes for different state-action pairs (Q-sets) remains a challenging problem [Reymond and Nowe, 2019]. Furthermore, even when the agent can learn a set of optimal values for each action, questions persist about how to effectively order actions during the training phase as these are Pareto incomparable [Felten et al., 2022]. Another technique, based on decomposition, allows scaling MORL algorithms to larger dimensions and real-world problems. This technique is presented in the next section.

2.3.4 Decomposition-Based Methods

Similar to decomposition in MOO (Section 2.2.5), it is also possible to decompose an MORL problem into several single-objective RL problems via the usage a scalarization function. The utilization of decomposition techniques offers several advantages for MORL. Firstly, by scalarizing rewards with different weights, these algorithms can often leverage single-objective RL techniques to learn multiple optimal policies. This allows multi-objective reinforcement learning based on decomposition (MORL/D) to directly benefit from advancements in RL research. Moreover, many algorithms from single-objective RL allow using function approximations such as DNNs and thus scale to larger dimensions and real-world applications [Mnih et al., 2015, Schulman et al., 2017]. Finally, scalarization provides a method for ordering the evaluations of potentially Pareto incomparable actions during the training phase, enabling the selection of actions in a greedy manner based on certain weights.

2.3.4.1 Example: Multi-policy Scalarized Q-Learning

In this trend, in the same vein as single solution-based MOO, the most straightforward algorithm, which is often called vanilla outer loop, proposes to sequentially train single-objective RL with different weights applied in

Algorithm 2.5 Multi-policy scalarized Q-Learning (MPQL). Also called vanilla outer loop.

Input: Stopping criterion *stop*, Environment *env*, Number of desired policies n.
Output: A set of trained policies Π.

1: $\Pi = \emptyset$
2: **for** n desired policies **do**
3: $\mathbf{w}$ = Generate weight vector
4: $q = Initialize()$
5: **while** $\neg stop$ **do** ▷ This is a traditional RL loop
6: $s = env.reset(), done = False$
7: **while** $\neg done$ **do**
8: $a = Sample(q)$
9: $s', \mathbf{r}, done = env.step(a)$
10: $q(s,a) = q(s,a) + \alpha\left(g(\mathbf{r},\mathbf{w}) + \gamma \max_{a' \in \mathcal{A}} q(s',a') - q(s,a)\right)$
11: $s = s'$
12: **end while**
13: **end while**
14: $\Pi = \Pi \cup q$
15: **end for**
16: **return** Π

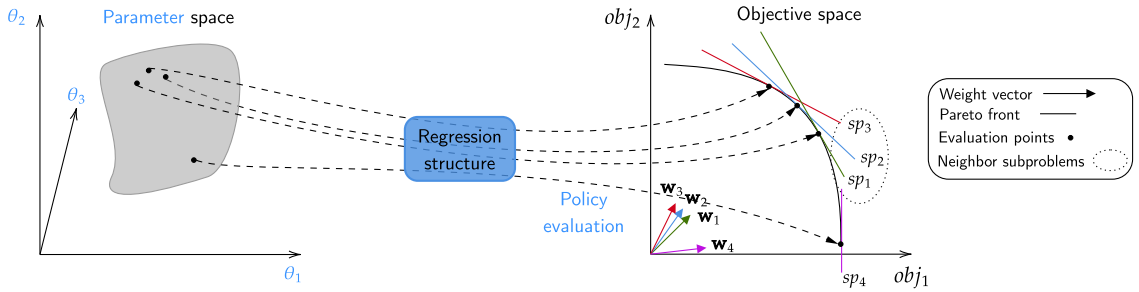


FIGURE 2.11: The decomposition idea applied to MORL. Blue parts emphasize the parts coming from RL, while black parts come from MOO. The optimization is looking for the best parameters for the regression structure to generate good policies. The idea of neighbor policies is that policies having similar parameters should lead to close evaluations.

the scalarization to target different parts of the objective space [Roijers and Whiteson, 2017]. For example, let us consider Algorithm 2.5: this algorithm modifies Q-learning [Watkins, 1989] by generating different weight vectors to target different areas of the objective space (line 3), then runs an RL training for each of these weights (lines 5–13). Regarding the Bellman update, scalarized Q-Learning adapts Equation 2.5 by transforming the reward vectors into scalars (line 10). Note that multiple policies targeting different areas of the objective space are stored in a set of policies Π, that is returned at the end of the training.

When compared to single-objective RL, this naive approach requires significantly more samples from the environment to compute various policies. Hence, the overarching goal of MORL/D methods is to perform better than this approach. Fortunately, there is a promising opportunity to leverage techniques from RL and MOO/D to improve the performance of MORL/D algorithms (Figure 2.11). Despite being a relatively recent area of study, several MORL works have already incorporated such improvements. Some ideas directly transposable from RL and MOO/D exist, while others demand specific adaptations and innovative approaches. Yet, despite using similar concepts, no study before ours clearly identified the links between these three fields of study. Figure 2.12 visually highlights the design choices and relations between the studied fields. The following parts delve into some of these choices in more depth, accompanied by examples from existing MORL literature that employ these techniques. Appendix A contains a table summarizing the classification of existing MORL works into our taxonomy.

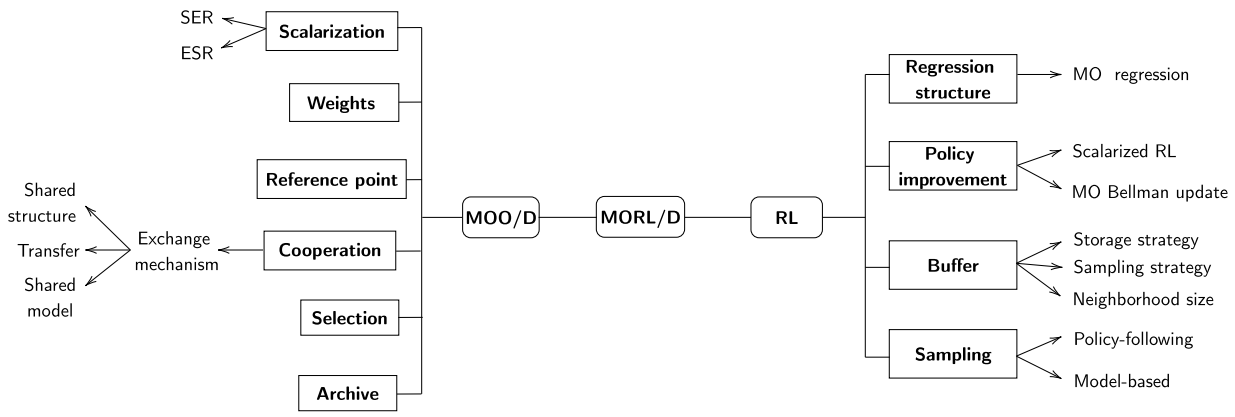


FIGURE 2.12: The Multi-Objective Reinforcement Learning based on Decomposition (MORL/D) taxonomy. Some traits from MOO/D and RL are directly applicable to this technique. Yet some alterations tailored for MORL have been published and are expanded in the figure.

2.3.4.2 Common Design Choices with RL

Existing techniques from RL can be adapted and applied directly in MORL/D. However, due to the unique multi-objective setting, certain components and aspects of these techniques may need to be modified to accommodate the specific requirements of MORL. This section describes the design choices and adaptations that originate from the field of RL but are relevant when applied in the context of MORL/D.

Regression Structure. As in classical RL, the choice of whether to use a function approximation or a tabular representation depends on the problem the user wants to tackle. Moreover, such a structure could also be adapted to incorporate multi-objective aspects.

Multi-objective regression structure. Several authors propose to slightly modify the way information is learned and encoded. First, conditioned regression involves using weight vectors as input to the regression structure, next to the current state. This allows for learning multiple policies within a single regression structure, with the hope that the regression structure learns to generalize over the objective space [Abels et al., 2019, Alegre et al., 2023, Lu et al., 2023]. Moreover, it is also possible to vectorize the value function estimator. For example, multi-objective DQN [Mossalam et al., 2016] outputs $|\mathcal{A}| \times m$ components, meaning that for each action, it predicts m values corresponding to each objective. Some results suggest that these vectorized value function structures may be more efficient than the scalarized ones [Abels et al., 2019], possibly because the regression structure does not need to capture the scalarization being used when maintaining vectorized estimators.

Policy Evaluation and Improvement. In RL, a crucial aspect is the policy improvement step, which typically relies on Bellman update equations. However, standard Bellman updates are designed to handle scalar rewards, whereas MORL deals with vectorized rewards. To adapt the Bellman update to multi-objective settings, various techniques and approaches have been proposed in the MORL literature.

• *Scalarized update.* The simplest approach, called scalarized update, is to use the scalarization function and rely on the Bellman updates from single-objective RL [Mossalam et al., 2016, Chen et al., 2020, Xu et al., 2020a]. The main advantage of this approach is that one can rely on the vast RL literature. This is the approach used in MPQL (Algorithm 2.5).

• *Multi-objective Bellman update.* Other approaches propose to modify the regression function to include more multi-objective aspects. In some cases, such as when relying on a conditioned regression, it makes more sense to optimize the predictions towards a good representation of the full PF. In this sense, Yang et al. [2019] proposes a Bellman update called *Envelope*, which optimizes not only across actions for each state but also for multiple weights over a space of preferences $\mathcal{W}$. It relies on the weighted sum scalarization g^{ws} and stores the Q-values as vectors. The update given an experience tuple $(s_t, a_t, \mathbf{r}_t, s_{t+1})$ and weight vector $\mathbf{w}$ is defined as follows:

$$\mathbf{q}(s_t, a_t, \mathbf{w}) \leftarrow \mathbf{q}(s_t, a_t, \mathbf{w}) + \alpha\left(\mathbf{r}_t + \gamma\left(\arg_{\mathbf{q}} \max_{a' \in \mathcal{A}, \mathbf{w}' \in \mathcal{W}} g^{\text{ws}}\big(\mathbf{q}(s_{t+1}, a', \mathbf{w}'), \mathbf{w}\big)\right) - \mathbf{q}(s_t, a_t, \mathbf{w})\right), \tag{2.16}$$

where the $\arg_{\mathbf{q}} \max_{a' \in \mathcal{A}, \mathbf{w}' \in \mathcal{W}}$ operator returns the $\mathbf{q}$ vector maximizing over all actions and weights.

Buffer Strategy. While experience buffers are common in RL to enhance learning efficiency, their utilization in MORL differs somewhat. In MORL, the agent aims to learn multiple policies simultaneously, which can impact how experiences are stored and shared. Indeed, it may be possible to improve the sample efficiency of MORL algorithms by using experiences sampled by one policy to improve other policies. Hence, the choice of how to manage experience buffers in MORL is influenced by the need to support the learning of multiple policies.

• *Replacement.* The classic way to organize an experience buffer is to store experiences based on their recency, where old experiences are discarded for new ones, also called first-in-first-out (FIFO). However, the problem with this kind of reasoning in MORL is that the sequence of experiences sampled by policies on different ends of the PF will probably be very different. Hence, the stored information may not be very useful to improve some policies. Thus, this replacement criterion of the buffer could be changed to include multi-objective criteria. Instead of replacing all the experiences in the buffer at each iteration, some elected experiences could stay in the buffer for various iterations. This allows for keeping a diverse set of experiences in the buffer. For example, Abels et al. [2019] proposes to store the return of a sequence of experiences as a signature for the sequence in the replay buffer, and a crowding distance operator similar to NSGA-II's [Deb et al., 2002] is used to keep the diversity of experiences in the buffer.

• *Selection.* Independently, the way to select which experience to use from the buffer to improve a policy can also include multi-objective criteria. The simplest sampling strategy is to pick experiences from the buffer uniformly. However, this strategy might select a lot of experiences leading to poor learning. As in classical RL, one way to improve the quality of these samples is to rely on priorities, similar to prioritized experience replay (PER) [Schaul et al., 2016]. Notably, this technique has been adapted to MORL settings, where each policy keeps its own priority based on its weights [Abels et al., 2019, Alegre et al., 2023].

• *Neighborhood size.* Instead of having either one buffer for all policies or one buffer per policy, an intermediate granularity of buffers could be shared between neighboring policies. In this way, the experiences in the buffer should be collected by policies that are close to the one being updated. The only existing work that has been found to address this idea is an ablation study presented in [Chen et al., 2020].

Sampling Strategy. As previously mentioned, the goal of an MORL agent is to learn multiple policies and is generally applied in an offline setting. Thus, in the training stage, there is no such thing as best action for each state, since these can be Pareto incomparable. In addition, various policies will probably lead to the exploration of different areas in the environment. Thus, adapting RL sampling strategies might be beneficial in such cases.

• *Policy following.* The straightforward adaptation of the RL sampling strategy to MORL is the one implemented in Algorithm 2.5. At each iteration, the agent chooses one policy to follow (associated with a given weight vector) and samples the environment according to this policy. For example, for a policy associated with a weight vector $\mathbf{w}$, Envelope Q-learning [Yang et al., 2019] relies on the scalarized Q-values to choose its actions, *i.e.*, $a^* = \arg\max_{a \in \mathcal{A}} g^{\text{ws}}(\mathbf{q}(s, a, \mathbf{w}), \mathbf{w})$. Additionally, when using a population-based algorithm, it is also possible to use multiple policies in parallel to fill the buffer at each iteration [Chen et al., 2020].

• *Model-based.* Another approach consists in constructing a model of the environment to systematically sample different areas [Moerland et al., 2023]. This allows for a more global sampling strategy. In this fashion, Felten et al. [2022] proposes to rely on metaheuristics to control the exploration of the agent and to use multi-objective metrics for the exploitation. This contribution is detailed in Section 3.1.

2.3.4.3 Common Design Choices with MOO/D

Most of the building blocks previously identified in MOO/D can immediately be applied to MORL/D. Indeed, weight vectors or reference points generation and adaptation schemes, Pareto archive, and individual selection mechanisms can be applied straightforwardly. However, some building blocks require particular attention; these are discussed in more detail below.

Scalarization. In RL, the evaluation of a policy comes from multiple rewards, collected over a sequence of decisions made by the agent. To reduce these to a scalar for evaluation, RL usually relies on the expectation of the discounted sum of rewards (Equation 2.2). In MORL/D, to have a total ordering of Pareto incomparable actions or to rely on single-objective policy improvements, a scalarization function is used. To transform this sequence of reward vectors into scalars for evaluation in MORL, the scalarization function can be applied before or after the expectation operator. Hence, the algorithm can optimize for the Expected Scalarized Return (**ESR**), or the Scalarized Expected Return (**SER**).

When using a linear scalarization, both settings are equivalent. However, the ESR and SER settings lead to different optimal policies when the scalarization is not linear [Roijers et al., 2018]. Formally:

$$\pi^*_{\text{SER}} = \arg\max_{\pi} g\left(\mathbb{E}_{a_t \sim \pi(s_t)}\left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(s_t, a_t, s_{t+1}) \mid s_0\right]\right)$$
$$\neq$$
$$\pi^*_{\text{ESR}} = \arg\max_{\pi} \mathbb{E}_{a_t \sim \pi(s_t)}\left[g\left(\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(s_t, a_t, s_{s+1})\right) \mid s_0\right].$$

Essentially, the choice between ESR and SER depends on the criticality of the application when executing the resulting policy. ESR, where scalarization is applied before the expectation, is suitable for critical applications like cancer detection, as every episodic return is crucial. In contrast, SER applies scalarization on the average return, making it suitable for repetitive applications, such as investments.

Additionally, when the learned policies are stochastic in the SER setting, the concave part of the PF is dominated by a stochastic mixture of policies [Vamplew et al., 2009]. This can be achieved, for instance, by employing one policy for one episode and another for the subsequent episode (see Figure 2.13). Hence, in such cases, one can restrict to using linear scalarization.

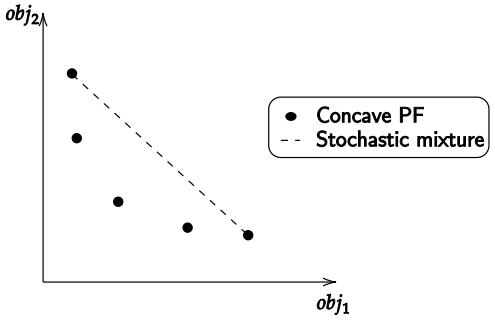


FIGURE 2.13: On average, policies resulting from a stochastic mix of deterministic policies dominate the policies in the concave part of the Pareto front.

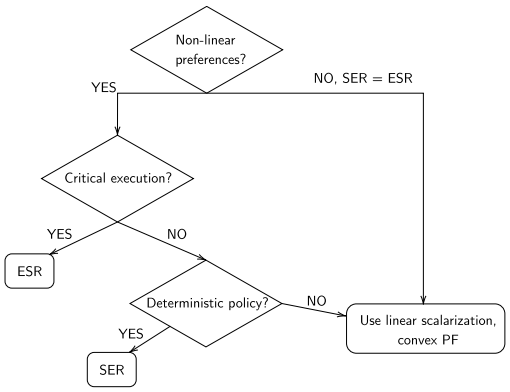


FIGURE 2.14: ESR vs. SER decision process.

Given these observations, capturing policies in the concave parts of the PF is only relevant when the user has a non-linear utility and aims to learn policies under the ESR criterion or deterministic policies under the SER criterion. Figure 2.14 summarizes the decision process to choose between ESR and SER settings in MORL. It is worth noting that in the literature most of the published algorithms optimize for the SER setting or under linear scalarization, leaving ESR understudied [Hayes et al., 2022, Röpke et al., 2023].

Finally, it is also important to note that optimal policies for non-linear scalarization may require conditioning not only on the state but also on the accumulated reward. This is because non-linear scalarization may break the additive property of the MOMDPs [Geibel, 2006, Roijers et al., 2018].

Cooperation. To improve the sample efficiency compared to the single solution approach (vanilla outer loop, Algorithm 2.5), various MORL works rely on cooperation mechanisms where the information gathered by a policy is shared with other policies. It is worth noting that RL policies are often over-parameterized, and thus, policies having very different parameters may lead to close evaluation points in the objective space. However, in MORL, these parameters are often kept similar. Usually, this is enforced by the fact that policies are initialized to be very close to each other (*e.g.*, via transfer learning), or their parameters are kept close to each other via the cooperation mechanisms.

While the neighborhood definitions and exchange triggers from MOO/D can be readily transferred to MORL/D, the knowledge exchange mechanism in MORL/D requires specific attention. This is because policies in MORL/D are typically encoded in regression structures or tables, which is different from the decision variable encoding in MOO/D. This distinction between MOO/D and MORL/D significantly impacts the design of information

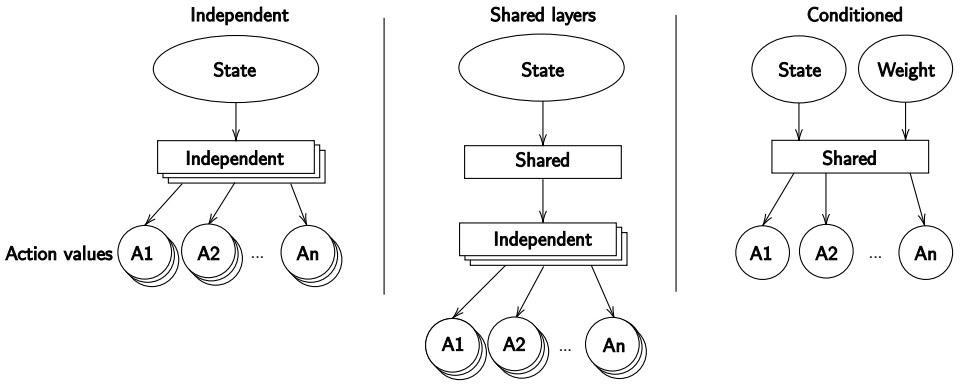


FIGURE 2.15: Shared regression structure schemes. Independent policies and conditioned regression lie on both ends of the shared regression spectrum. Independent policies do not share any parameter, whereas conditioned regression allows encoding multiple policies into a single regression structure. Shared layers lie in the middle, sharing part of the network parameters while leaving some to be independent.

exchange techniques in the context of multi-objective reinforcement learning. Based on the surveyed articles in the MORL literature, we identify three ways to exchange information between neighbors.

• *Shared regression structure.* While vanilla outer loop typically relies on completely independent policies (Figure 2.15, left), some works propose to share information between distinct policies by directly sharing part (or all) of the regression structure. This way, part of the parameters are shared, allowing to learn fewer parameters but also to share gathered experiences with multiple policies at the same time. In this fashion, Chen et al. [2020] proposes to share some base layers between multiple DNNs representing the policies (Figure 2.15, middle). More extreme cases, such as Abels et al. [2019], propose using only one regression structure by conditioning the input on the weights (Figure 2.15, right). This approach, discussed in Section 2.3.4.2, also works for regression trees [Castelletti et al., 2013], hence we refer to it as conditioned regression (CR).[4] While it may provide faster convergence to new points in the PF, this technique may forget previously trained policies unless tailored mechanisms are deployed [Abels et al., 2019].

Finally, similar to hindsight experience replay [Andrychowicz et al., 2017], conditioning the networks on the weights also allows the augmentation of experience data by sampling different weights for the same experience tuple [Yang et al., 2019].

• *Transfer.* Transfer learning is a common machine learning technique that involves utilizing parameters from a previously trained regression structure to initiate the training of a new regression. In the context of MORL, transfer learning has been applied to various algorithms, allowing the training of a new policy to start from the parameters of the closest trained neighbor policy. This approach has demonstrated improved performance in some MORL algorithms [Natarajan and Tadepalli, 2005, Roijers et al., 2015b].

• *Shared model.* Another way to exchange information between policies is to share a model of the environment. The idea is that the environment dynamics, *i.e.*, p and $\mathbf{r}$, are components that need to be estimated, but are the same for each policy that needs to be learned.

In this fashion, the simplest way to share a model is to use experiences sampled from the environment by one policy to apply Bellman updates to neighbor policies. Such a technique can drastically improve the sample efficiency of MORL methods, yet it is restricted to the use of an underlying off-policy learning algorithm. This idea is similar to sharing the search memory in MOO/D and is usually achieved by sharing the experience buffer between multiple policies in MORL [Abels et al., 2019, Chen et al., 2020].

[4] This is similar to what is termed *parameter sharing* in multi-agent RL, *e.g.*, Yu et al. [2022].

Another technique proposes to use model-based reinforcement learning to learn the dynamics and rewards of the environment in a surrogate model, and to use the learned model to generate samples to learn policies [Wiering et al., 2014, Alegre et al., 2023]. Such an approach also improves sample efficiency in MORL algorithms. This setting is very similar to what is called "same dynamics with different rewards" in model-based RL [Moerland et al., 2023].

2.3.5 Use Cases Scenarios

Due to its relatively recent emergence in research, MORL has not yet found widespread deployment in real-world applications. However, the field is progressing, and benchmark problems have been recently introduced to facilitate further research [Alegre et al., 2022, Felten et al., 2023a]. It is anticipated that MORL can be applied in similar domains as traditional RL [Vamplew et al., 2022]. Indeed, real-world problems typically involve optimizing multiple objectives. In the field of **robotics**, for instance, agents often aim to maximize task-specific rewards while minimizing energy consumption or component wear [Alegre et al., 2022]. More recently, in the domain of **large language models** (LLM), RL has been utilized for fine-tuning based on human feedback. In these scenarios, LLMs are customized to be more specialized using various reward models derived from human preference datasets [Ramé et al., 2023, Jang et al., 2023]. This may involve the LLM seeking to maximize both accuracy and fairness. Additional instances of applications could include **real-time scheduling**, such as task assignment on a high-performance computer or dynamic fuzzy job-shop [Chen et al., 2023], **optimization of molecules** [Zhou et al., 2019], or **network optimization**, where controllers on antennas would balance between latency, throughput, and energy consumption. Yet, in practical applications, rewards are typically linearly scalarized *a priori*, potentially leading to suboptimal regions in the Pareto front (Figure 2.9).

2.3.6 Summary

While the initial two sections provided background information on RL and MOO, this section presented the domain of multi-objective RL. In this setting, the objective is to learn a set of Pareto optimal policies in MOMDPs. Various solving techniques from the literature have been presented and clearly linked to existing ones coming from RL and MOO using a newly introduced taxonomy. These techniques usually make MORL algorithms more sample-efficient than running multiple times a scalarized single-objective RL algorithm.

However, challenges arise in scenarios where multiple agents operate within a shared environment, such as in a swarm of autonomous drones. Addressing this concern, the following section introduces an extension of multi-objective RL tailored for multiple cooperative agents.

2.4 Multi-Objective Multi-Agent Reinforcement Learning

Multi-objective multi-agent reinforcement learning (MOMARL) is a relatively new field of research that extends MORL to settings where multiple agents enact and influence the environment, as depicted in Figure 2.16. These agents can cooperate, compete, or operate with a mix of both strategies. Moreover, agents can act jointly at each step or take turns, similar to a board game.

Multiple settings arise when augmenting the multi-agent problems with multiple objectives, depending on whether the agents are collaborative or competitive, whether they receive the same or different rewards, and whether they have the same preferences about these rewards. Figure 2.17 summarizes these settings and

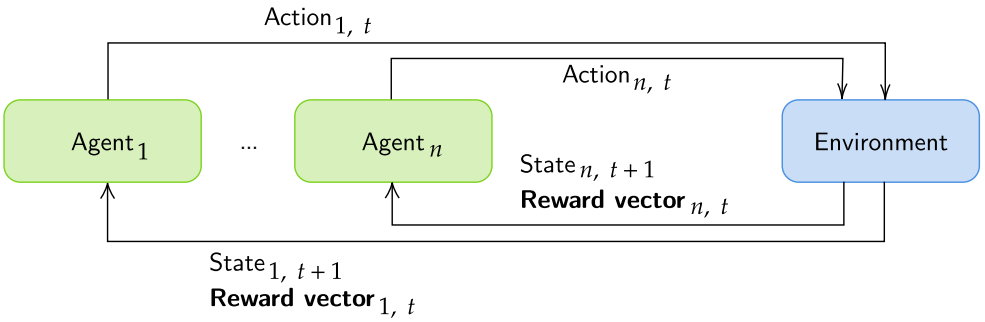


FIGURE 2.16: Multi-objective multi-agent reinforcement learning agents-environment interactions.

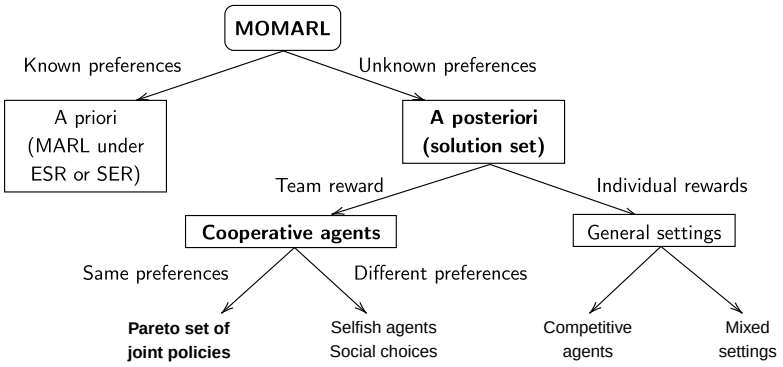


FIGURE 2.17: MOMARL settings. The settings explored in this thesis are highlighted in bold. A thorough review of all settings can be found in Rădulescu et al. [2020].

highlights the settings we focus on in this thesis, that is cooperative agents with equal but unknown preferences. Indeed, we focused on cooperative problems for swarms of drones which we assume to be homogeneous, care only for a global team reward, and finally be deployed by a single entity, hence sharing the same preferences. A thorough overview of all potential settings and solution concepts is discussed in the seminal work of Rădulescu et al. [2020].

2.4.1 Formation Definitions and Notations

The most straightforward extension of an MOMDP (Section 2.3.1) to multi-agent cooperative settings is a multi-objective multi-agent Markov decision process (MOMMDP) [Rădulescu et al., 2020]. Formally, an MOMMDP is defined by a tuple $(n, \mathcal{S}^n, \mathcal{A}^n, \mathbf{r}, p, \gamma, \mu_0)$ and extends the MOMDP where:

- n is the number of agents;
- $\mathcal{S}^n = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ are the states the agents can perceive, where $\mathcal{S}_i$ is the state space of agent i;
- $\mathcal{A}^n = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is the space of joint actions, where $\mathcal{A}_i$ is the action space of agent i;
- $\mathbf{r}\colon \mathcal{S}^n \times \mathcal{A}^n \times \mathcal{S}^n \mapsto \mathbb{R}^m$ is the team reward function.[5]

At every timestep, the environment typically emits a joint state $\mathbf{s} = (s_1, \ldots, s_n)$, where $\mathbf{s} \in \mathcal{S}^n$. Depending on the system, the agents can fully observe this state, or can only partially observe it. This latter setting is defined as a multi-objective decentralized partially observable MDP (MODec-POMDP, extending Dec-POMDPs to MO

[5]Note that in more general settings, each agent receives a different reward and there are usually one reward function per agent.

settings [Oliehoek and Amato, 2016]), which although more general makes notations heavier and is orthogonal to the multi-agent or multi-objective dimensions. Thus, we chose to leave partial observability aside in this thesis. Finally, it is worth emphasizing that the reward received by the agents depends on the joint state and joint actions taken by all the agents in the environment and not just on their own [Rădulescu et al., 2020].

2.4.2 Solution concepts

In the studied settings, the solutions concepts of an MOMMDP for agents having unknown but same preferences are very similar to MORL, *i.e.*, a Pareto set of joint policies and its associated Pareto front. This setting is referred to as *team reward team utility* in the taxonomy of Rădulescu et al. [2020]. A joint policy, denoted $\boldsymbol{\pi}$, is defined by a set of policies, one for controlling each agent, *e.g.*,

$$\boldsymbol{\pi} = (\pi_1, \dots, \pi_n).$$

Naturally, in cooperative settings with team reward, the multi-objective multi-agent value function has a similar shape (*i.e.*, vectorial) as in single-agent settings:

$$\mathbf{v}^{\boldsymbol{\pi}} = \mathbb{E}_{\mathbf{a}_t \sim \boldsymbol{\pi}(s_t)} \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \mid \mu_0 \right].$$

Notice the changes induced by the multi-agent settings: the value depends on a joint policy, and the reward function depends on joint states and joint actions. Finally, the solution concepts (PS and PF) from Equation 2.14 and 2.15 can be trivially extended based on these definitions.

2.4.3 Multi-Agent Solving Methods

Before diving into MOMARL settings, this section provides an overview on the current solving methods used in multi-agent RL.

Centralized learners. One simple way to deal with multi-agent problems is to simply learn all policies within a single centralized learner that has a global view of the environment and outputs joint actions. However, the action space rapidly becomes unwieldy when scaling the number of agents [Schroeder de Witt, 2021]. Furthermore, deploying centralized policies in decentralized settings presents challenges, limiting the application of such techniques to scenarios with robust communication and susceptibility to single points of failure. Hence, such methods are rarely used in practice.

Independent learners. On the other end of the spectrum lies independent learners, where each agent learns its own policy, as in Independent Q-Learning (IQL) [Tan, 1993]. In such settings, determining whether a particular agent's action or the actions of others were beneficial or detrimental poses challenges, known as the *multi-agent credit assignment problem* [Foerster et al., 2018, Schroeder de Witt, 2021]. Furthermore, evaluating an agent's policy also depends on the actions of other agents, *i.e.*, their policies. This renders the problem *non-stationary*, as multiple agents learn and adjust their policies simultaneously, making multi-agent settings notoriously difficult to address [Foerster et al., 2017], and ultimately making such methods provide no convergence guarantee even in tabular settings [Schroeder de Witt, 2021]. Yet in practice, independent learners

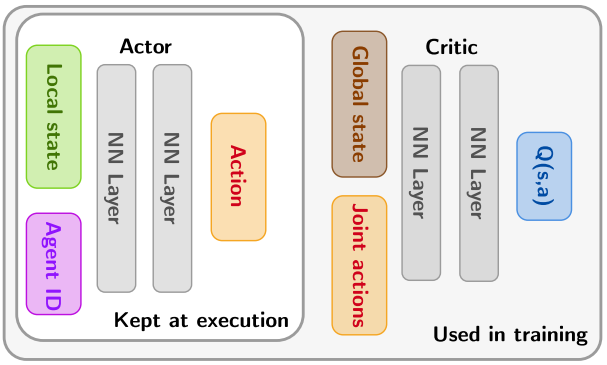


FIGURE 2.18: Neural architecture of MARL actor-critic algorithms applying the centralized training decentralized execution scheme with parameter sharing.

often give surprisingly strong baselines for comparisons and are thus often used in research papers. Finally, independent learners make the number of parameters to learn scale with the number of agents, which is often impractical for real-world applications.

CTDE methods. A promising idea to solve the multi-agent challenge lies in between the two last paradigms by relying on a centralized training with decentralized execution (CTDE) scheme [Oliehoek et al., 2008]. These CTDE methods usually adopt multiple regression structures. A centralized network (in general a critic, Figure 2.18, right), whose task is to approximate the value of some joint state-action pairs ($Q^{\boldsymbol{\pi}}(\mathbf{s}, \mathbf{a})$), allows to perform temporal-difference learning. This network, which is only used while training, can thus learn centralized critic that takes global information as input. Relying on such global information typically eases training by providing better information and mitigates the credit assignment problem by having a global credit assignment while training. Moreover, conditioning the critic on joint actions allows mitigating the non-stationarity problem [Lowe et al., 2020]. The policies (Figure 2.18, left), which determine the agents' actions, must still use their local observations as input since it is the only information available at the execution phase when deployed in a decentralized manner.

Parameter sharing. MARL algorithms can also avoid the explosion of parameters to learn by sharing the policies' parameters among the agents and using only one structure to learn multiple policies [Schroeder de Witt, 2021]. This is usually done by concatenating the agent's id to the input of the policy, see Figure 2.18, left. This also allows learning of the environment dynamics for multiple agents at once, effectively improving sample efficiency. Note the similarity with conditioned regression in MORL (Figure 2.15).

Because of their advantages (*i.e.*, mitigating the non-stationarity and credit assignment problems, efficient learning, and limitation of learning parameters), the current state of the art in MARL often relies on the two latter techniques, that is CTDE and parameter sharing, *e.g.*, Rashid et al. [2020], Yu et al. [2022]. Using such an architecture, single-agent RL algorithms can easily be modified to solve multi-agent problems, see for example multi-agent deep deterministic policy gradient (MADDPG) [Lowe et al., 2020] or multi-agent proximal policy optimization (MAPPO) [Yu et al., 2022].

2.4.4 Multi-Objective Multi-Agent Solving Methods

MOMARL, being a relatively nascent research domain, has seen a limited number of publications thus far, with most methods employing scalarization a priori, as exemplified by Mannion et al. [2016]. A limited number of works have proposed methodologies capable of learning a Pareto set of joint policies within the studied contexts. These endeavors encounter several challenges:

1. Parameter explosion: The complexity escalates significantly due to the multi-agent and multi-objective dimensions, leading to difficulties in managing the growing number of parameters to learn.
2. Multi-agent challenges: These encompass the credit assignment problem, where determining the contribution of individual agents to the overall performance becomes intricate in a multi-agent setting, alongside the non-stationarity issues posed by updates of multiple policies at once.
3. Multi-objective challenges: Being able to learn a set of Pareto optimal policies to present trade-offs to the user.

Given the first challenge, it is evident that tabular-based methods may not be viable for real-world applications due to scalability issues. This poses the necessity of function approximation, such as DNNs, as acknowledged in MARL [Schroeder de Witt, 2021]. We would like to remember that Pareto-based methods in MORL are not yet applicable under function approximation. Therefore, decomposition stands out as a promising approach for developing MOMARL algorithms. Leveraging decomposition also allows us to tap into the existing knowledge base of MARL, thereby addressing the second challenge through established multi-agent methodologies. The decompositional aspects, on their side, allow learning various policies leading to different trade-offs, similar to what has been presented in MORL/D but relying on a MARL method instead of single-agent RL, see our taxonomy presented in Section 2.3.4.

In the existing literature, MO-MIX [Hu et al., 2023] is a notable contribution capable of learning a PS of joint policies. MO-MIX integrates elements of QMIX [Rashid et al., 2020] (a CTDE method) and decomposition: linear scalarization, conditioned regression as cooperation pattern, multi-objective regression, an update rule inspired by Envelope (Equation 2.16), and generates the weight vectors adaptively by computing probabilities based on how many optimal points are in a given subregion of the objective space, effectively giving more probability to generate weight vectors targeting sparse regions. Despite not employing parameter sharing, MO-MIX effectively manages the multi-objective dimension by employing conditioned regression, thus reducing the parameter overhead for each agent by learning multiple trade-off policies within a unified network structure. The multi-agent challenges are effectively addressed by the QMIX part of the algorithm, while the decompositional aspects handle the learning of a PS of joint policies.

2.4.5 Use cases scenarios

Despite its nascent status, MOMARL has garnered attention in scholarly discourse, with numerous works exploring its theoretical underpinnings and potential applications [Mannion et al., 2016, Roijers and Whiteson, 2017, Rădulescu et al., 2020]. Currently, the majority of published MOMARL applications have resorted to either scalarizing the problem *a priori* and employing conventional MARL algorithms or centralizing the agent's decision-making process, drawing from MORL frameworks. These approaches have found utility in addressing challenges inherent to **dynamic scheduling and planning tasks**, where the size of action space precludes effective centralized MORL solutions, as observed in domains such as infrastructure management planning [Leroy

et al., 2023] and multicore processor task scheduling [Ahmad et al., 2008]. Furthermore, MOMARL has been used in **smart grids**, where it controls the timing of appliance operation to minimize costs, reduce peak power demand, and accomplish tasks efficiently [Lu et al., 2022b]. Additionally, MOMARL finds relevance in **traffic signal control**, wherein each traffic light constitutes an autonomous agent, with objectives including average waiting time and maximum queue length at intersections [Houli et al., 2010], or taxation policy, making a trade-off between producitivity and equality [Zheng et al., 2022]. Moreover, the domain of **drone swarming** presents fertile ground for MOMARL, where multiple drones collaborate to achieve collective goals, such as zone coverage optimization while minimizing battery consumption. Section 6.2 of our thesis aims to address precisely these challenges, illustrating the practical relevance and potential of MOMARL in real-world scenarios.

2.5 Conclusion

This section lays out the foundational groundwork, including background information, essential notations, and solution concepts essential to the diverse settings explored within this thesis. First, Section 2.1 introduced the principles of reinforcement learning, delineating the pursuit of maximizing returns through agent-environment interactions. Following this, Section 2.2 introduced the realm of multi-objective optimization, where methodologies aim at finding trade-offs among multiple objectives. Building upon these fundamental domains, Section 2.3 presented the domain of multi-objective reinforcement learning, where agents contend with conflicting objectives, striving to learn diverse Pareto optimal policies. This area of research relies on insights from the preceding fields, which have been exposed clearly through the introduction of our taxonomy. Lastly, Section 2.4 extended MORL into domains featuring multiple agents interacting within a single environment, thus making a novel research field named multi-objective multi-agent reinforcement learning. This combination of multi-agent RL and multi-objective RL introduces significant complexity, yet its necessity is underscored by existing applications.

Subsequent sections of the thesis present our contributions within these domains. Part II illustrates our contributions in MORL, including the enhancement of existing solution methodologies, and the provision of robust tools essential for MORL research. Then, Part III first presents our foundational contributions to MOMARL aimed at accelerating research in this burgeoning field. Then, we present a real-world instantiation of the methodologies discussed throughout the thesis, showcasing their applicability and efficacy in practical scenarios. Finally, Part IV concludes our work and discusses potential future research directions.

Published Work

The taxonomy and background on RL, MOO/D, and MORL/D discussed in this Sections 2.1–2.3 have been published in the Journal of Artificial Intelligence Research:

- Florian Felten, El-Ghazali Talbi, and Grégoire Danoy. Multi-Objective Reinforcement Learning Based on Decomposition: A Taxonomy and Framework. *Journal of Artificial Intelligence Research*, 79:679–723, February 2024a. ISSN 1076-9757. doi: 10.1613/jair.1.15702. URL https://www.jair.org/index.php/jair/article/view/15702

Part II

Single-agent Multi-Objective Reinforcement Learning

Life is made up of compromises.

— *Edith Wharton*

Chapter 3

Using Multi-Objective Optimization Techniques in MORL

Contents

As discussed earlier, existing MOO techniques can be readily applied to solve MORL problems. This chapter studies the usage of MOO techniques for Pareto-based MORL and Decomposition-based MORL.

Section 3.1 showcases for the first time the usage of metaheuristics, classical MOO techniques, to address the exploration-exploitation dilemma in Pareto-based MORL. Importantly, our findings demonstrate the superior performance of these methods compared to the commonly used ϵ-greedy approach in Pareto-based MORL.

In Section 3.2, we introduce a comprehensive framework for decomposition-based MORL algorithms, building upon the taxonomy outlined in Section 2.3.4. This framework, rooted in the rigorous analysis of RL, MOO, and MORL done earlier, facilitates the adaptation of techniques from fields having benefitted from greater attention than MORL to MORL itself. Notably, our framework enables the modification of specific algorithmic components to explore the impact of novel techniques. Our results demonstrate the versatility of the framework across various problem types, including concave and convex Pareto fronts, as well as continuous and discrete action and state spaces. Moreover, our initial implementation exhibits performance comparable to the current state of the art.

3.1 Metaheuristics-Based Exploration Strategies for Pareto-Based MORL

Section 2.3.3 presented Pareto-based MORL algorithms and Pareto Q-Learning (PQL) as an example. In this section, the question of how to choose which action to take at the learning phase given the current state and Q-sets, has been raised. The present section dives deeper into what matters regarding exploration and exploitation for Pareto-based MORL. Notably, it presents novel exploration methods, drawn from the optimization realm, and their application to PQL. The results show that the presented techniques consistently give better results than the current state-of-the-art exploration mechanisms.

As discussed in Section 2.1.3.4, the exploration-exploitation dilemma (EED) is a well-studied subject in RL and optimization. When looking for optima, there is a trade-off between looking in the neighborhood of good, known solutions (*exploitation*) and trying to find new solutions somewhere else in the search space (*exploration*). Too much exploitation would mean the algorithm could get stuck in local optimum while too much exploration would mean the algorithm potentially does not converge to any optimum. Decisions on such matters can significantly influence the performance of these algorithms. Notably, due to its critical importance, the EED has been thoroughly investigated in traditional single-objective RL settings [Amin et al., 2021]. Yet, the EED has rarely been studied in multi-policy MORL and there is limited insight into the effectiveness of these standard techniques in the MORL settings. In fact, to the best of our knowledge, no work has been dedicated to studying the exploration question in Pareto-based algorithms yet [Vamplew et al., 2017a].

This section thus studies the EED for Pareto-based MORL algorithms. It presents three novel exploration strategies for MORL inspired by the metaheuristics domain. Metaheuristics have a long history in addressing optimization problems characterized by expansive search spaces, often rendering the application of exact methods impractical. The primary objective of these methods is to guide the search process towards favorable regions within a reasonable timeframe [Talbi, 2009]. The three proposed strategies include pheromones-based, count-based, and tabu-based approaches. Grounded in the concept of repulsion from already visited states and actions, these strategies aim to systematically encourage the discovery of new solutions.

The rest of this section is organized as follows: Section 3.1.1 first presents a template algorithm for Pareto-based MORL derived from PQL (Algorithm 2.4), which allows to atomically replace exploration and exploitation

Algorithm 3.1 Metaheuristics based MORL template, similar to PQL (Algorithm 2.4).

Input: Stopping criterion *stop*, Environment *env*, Heuristic function *Heuristic*, Metaheuristic *Meta*, Metaheuristic update *UpdateMeta*, Q-sets learning mechanism *UpdateQsets*.
Output: The Q-sets $\mathcal{Q}$.

1: $\mathcal{Q} = \emptyset$
2: **while** $\neg stop$ **do**
3: $s = env.reset(), done = False$
4: **while** $\neg done$ **do**
5: $a = Meta(s, \mathcal{Q}, Heuristic)$
6: $s', \mathbf{r}, done = env.step(a)$
7: $\mathcal{Q} = UpdateQsets(\mathbf{r}, s, a, s', \mathcal{Q})$
8: $s = s'$
9: **end while**
10: $Meta = UpdateMeta(Meta)$
11: **end while**
12: **return** $\mathcal{Q}$

strategies. Section 3.1.2 and 3.1.3 respectively discuss the exploitation and exploration strategies for Pareto-based MORL in more detail. Section 3.1.4 discusses our experiments and presents a new episodic, deterministic benchmark environment, called Mirrored Deep Sea Treasure (MDST). Moreover, experiments are conducted on multiple MORL environments, showing the three proposed metaheuristics-based strategies outperform state-of-the-art techniques. Lastly, Section 3.1.5 summarizes these contributions.

3.1.1 Learning Algorithm

First, let us consider Algorithm 3.1 as a template Pareto-based learning algorithm, similar to Algorithm 2.4. It is modular enough to completely separate the learning from the exploration and exploitation parts in the training phase. The algorithm starts by initializing the $\mathcal{Q}$ sets (line 1). Then for a given optimization budget, it trains on the environment *env*. Given the current state s and current Q-sets $\mathcal{Q}$, the agent chooses the next action to perform using a combination of the heuristic value computed from the *Heuristic* function and metaheuristic decision-making from the *Meta* function (line 5). When the agent performs an action in the environment, the reward vector $\mathbf{r}$ and the next state are returned (line 6). Using this information, the learning algorithm is applied to update the $\mathcal{Q}$ sets (line 7). The state is then updated to choose the move at the next iteration (line 8). At the end of the episode, the metaheuristic can also be updated in certain cases (line 10).

This algorithm brings flexibility. As a matter of fact, it allows one to replace some of its components without the need to change the other ones, allowing one to perform ablative studies of various methods. The learning part (*UpdateQsets*) can be implemented using the work of Van Moffaert and Nowé [2014] or Ruiz-Montiel et al. [2017] (see Section 2.3.3). The *Heuristic* function embodies the exploitation part of the algorithm and is described in the next section. The metaheuristic part, *Meta* and *UpdateMeta*, which controls the exploration of the agent is described in Section 3.1.3.

3.1.2 Exploitation (*Heuristic*)

In Pareto-based MORL, Q-values become $\mathcal{Q}$ sets, *i.e.*, sets of vectors. This prevents the application of traditional operators to greedily choose an action, *e.g.*, arg max, or softmax. As mentioned in Section 2.2.4, performance indicators or Pareto dominance ranking functions can convert such sets into scalar values (called fitness in multi-objective EAs), then allowing to use the maximization operators. In this study, we use the hypervolume indicator to perform such scalarization because of its hybrid nature, *i.e.*, it gives an idea of quality in terms of

Algorithm 3.2 Tabu-based *Meta*

Input: Tabu list $\mathcal{T}$, Max tabu size τ, Current state s, Q-sets $\mathcal{Q}$, Heuristic function *Heuristic*.
Output: The next action to perform a.

1: $\mathcal{NT} = \{a \ : \ \forall a \in \mathcal{A} \mid (s, a) \notin \mathcal{T}\}$
2: $a = -1$
3: **if** $\mathcal{NT} = \emptyset$ **then**
4: $\quad a = Random(|\mathcal{A}|)$
5: **else**
6: $\quad H = \{Heuristic(\mathcal{Q}(s, a)) \ : \ \forall a \in \mathcal{NT}\}$
7: $\quad a = \arg\max_{a \in \mathcal{NT}} H(a)$
8: **end if**
9: $\mathcal{T} = \mathcal{T} + (s, a)$
10: **if** $|\mathcal{T}| > \tau$ **then**
11: $\quad \mathcal{T}.pop()$
12: **end if**
13: **return** a

convergence and diversity. This allows to heuristically choose actions associated with the best Q-sets. Obviously, the proposed method stays valid for other indicators such as cardinality or Pareto rank [Van Moffaert and Nowé, 2014].

3.1.3 Exploration (*Metaheuristic*)

Although it has become a standard in (MO)RL, the ϵ-greedy exploration strategy, which has been used in various Pareto-based MORL algorithms [Van Moffaert and Nowé, 2014, Ruiz-Montiel et al., 2017], could be harmed by its simplicity. It chooses the best action greedily with $1 - \epsilon$ probability and a random action in all other cases. It seems more information could be leveraged when exploring. Indeed, the algorithm could choose a non-greedy move based on a score assigned to each of them, as in the softmax exploration strategy. In addition, akin to "state-based exploration" [Moerland et al., 2023], the algorithm could keep a memory of moves that were already sampled in order to focus on less explored areas [Badia et al., 2020]. In a similar manner, this part of the thesis proposes three exploration strategies coming from the metaheuristics field: tabu-based, count-based, and pheromones-based. All these are based on the principle of *repulsion*: well-exploited zones should be less attractive to the behavioral policy, as to facilitate a better coverage of the state-action space. These strategies and their associated algorithms can be used as the metaheuristic part of Algorithm 3.1. The rest of this section describes each strategy in more detail.

Tabu-based. Tabu search is a well-known trajectory-based metaheuristic that has been widely used in various problems. The idea is to mark recently chosen actions in a given state as tabu, meaning they should not be considered the next time the agent encounters the same state. The implementation of the tabu-based *Meta* function for our problem is presented in Algorithm 3.2.

For every move choice, the agent first selects all the current moves considered as non-tabu from the list of possible moves (line 1). It then either selects a move randomly if no move is non-tabu (lines 3–4), or it computes a heuristic value from the $\mathcal{Q}$ set of each considered action using the exploitation mechanism (*e.g.*, hypervolume) (line 6). The best move is chosen greedily as the one having the largest heuristic value (line 7). The tabu list is then updated to avoid taking the same action the next time the agent encounters this state (line 9). This is the repulsive mechanism. To allow for the sampling of state-action pairs multiple times, addressing potential stochasticity in the environment dynamics, the tabu list is constrained by a parameter τ. After some time, the state-action pairs are removed from the tabu list, allowing one to resample this combination (lines 10–11).

Algorithm 3.3 Count-based *Meta*

Input: State-action counter n, Minimum hypervolume value min, Heuristic weight α, Count weight β, Current state s, Q-sets $\mathcal{Q}$, Heuristic function *Heuristic*.
Output: The next action to perform a.

1: $H = \{(a, Heuristic(\mathcal{Q}(s,a))) \ : \ \forall a \in \mathcal{A}\}$
2: $S = \{\frac{([\eta]_{min})^\alpha}{(n(s,a))^\beta} \ : \ \forall (a,\eta) \in H\}$
3: $a = \arg\max_{a \in \mathcal{A}} S(a)$
4: $n(s,a) = n(s,a) + 1$
5: **return** a

Algorithm 3.4 Pheromones-based *Meta*

Input: Pheromones P, Minimum hypervolume value m, Heuristic weight α, Pheromones weight β, Current state s, Q-sets $\mathcal{Q}$, Heuristic function *Heuristic*.
Output: The next action to perform a.

1: $H = \{(a, Heuristic(\mathcal{Q}(s,a))) \ : \ \forall a \in \mathcal{A}\}$
2: $S = \{\frac{([\eta]_m)^\alpha}{(P(s,a))^\beta} \ : \ \forall (a,\eta) \in \mathcal{A}\}$
3: $S(a) = \frac{S(a)}{\sum_{a' \in \mathcal{A}} S(a')}$
4: $a = Sample(\mathcal{A}, S)$
5: $P(s,a) = P(s,a) + 1$
6: **return** a

Algorithm 3.5 UpdatePheromones: *UpdateMeta* providing the evaporation system for Pheromones-based metaheuristics

Input: Pheromones P, Evaporation factor ρ.
Output: The updated pheromones UP.

1: $UP = \{\rho P(s,a) \ : \ \forall P(s,a) \in P\}$
2: **return** UP

Choosing the size of the tabu list depends on the size of the problem. No *UpdateMeta* function is needed in this case.

Count-based. This exploration strategy is commonly used in RL and optimization. The idea here is to reduce the score associated with state-action pairs according to the number of times these pairs have been sampled. Its implementation is listed in Algorithm 3.3.

The metaheuristic starts by computing the heuristic values of each possible move (line 1). Then, each value is min-clipped with a value $min > 0$ to avoid moves with a 0-valued heuristic. This value is then divided by the number of times the state-action pair has been selected, $n(s,a)$. α and β are hyperparameters allowing the control of the exploration-exploitation intensity depending on the problem at hand (line 2). Finally, the best action is chosen according to the ratio computed earlier, and the related count is updated (lines 3–5).

Our repulsing mechanism in this case relies on the count. Because $n(s,a)_t$ is unbounded, there always exists a time step t where the value of the greedy action according to the *Heuristic* function will be reduced as much as to choose another action, as all moves have strictly positive values due to min-clipping. This is why the actions are chosen deterministically, using the arg max operator, while keeping the guarantee of visiting all state-action pairs in an infinite horizon (and thus optimality guarantees of the learning algorithm). No *UpdateMeta* function is needed in this case either.

Pheromones-based. Inspired from the Ant-Q algorithm introduced in Gambardella and Dorigo [1995], the algorithm defined in Algorithm 3.4 uses the concept of repulsive pheromones as *Meta* function. In this case, the agent leaves a pheromone on the path it has taken. Similar to the count-based approach, these pheromones are used to reduce the score associated with state-action pairs that have been chosen recently. The difference with the previous approach is that pheromones evaporate after each episode (see Algorithm 3.5), as to increase the probability for an agent to take an already explored path and resample it.

As in Algorithm 3.3, the agent starts by applying the heuristic function to the Q-sets (line 1). Then, this value is divided by the amount of pheromones assigned to that state-action pair, noted $P(s, a)$. The min-clipping approach is used here as well to ensure all heuristic values are greater than zero (line 2). In this case, the pheromones P are bounded because of the evaporation mechanism presented in Algorithm 3.5. Therefore, there might exist a heuristic score for an action a such that $S(a)_t > S(a')_t \; \forall a' \in \mathcal{A} \setminus \{a\} \; \forall t$ in a given state. Hence, this action might always be chosen, leading to never exploring the other ones. This is why the actions' scores are first normalized before the metaheuristic samples the action according to the generated distribution (lines 3–4). The pheromones are then updated (line 5). Finally, the *UpdateMeta* is used for the evaporation mechanism as shown in Algorithm 3.5. This algorithm multiplies all pheromones by an evaporation factor $\rho \in [0, 1)$.

3.1.4 Experiments

This section presents how the performances of the proposed strategies have been assessed as well as the results obtained. We performed experiments on various MORL environments and compared them against state-of-the-art ϵ-greedy-based methods typically used in Pareto-based MORL works such as Van Moffaert and Nowé [2014], which employs a decaying ϵ-greedy, and Ruiz-Montiel et al. [2017], which uses a constant ϵ-greedy.

Metrics. The metrics used for comparison are common metrics from the MORL and MOO literature: hypervolume, IGD, and cardinality (see Section 2.2.3). All metrics have been aggregated over 40 runs and reported as learning curves. Additionally, the results are also reported in tables containing the mean hypervolume and standard deviation of the hypervolume of the vectors seen from the starting state of the environments ($\mathcal{F}(s_0)$ in Section 2.3.3) every 500 training episodes. Cells are colored using a linear gradient; the greener, the better. Finally, to study the statistical difference between the methods, both ANOVA and Tukey's tests are performed on the hypervolume metrics of each newly introduced metaheuristic and the two state-of-the-art methods, with $\alpha = 0.05$. In the results tables, the values for which the hypervolume of the proposed metaheuristic-based strategy is better than the state-of-the-art ones with statistical confidence (*i.e.*, ANOVA, and Tukey tests positive) are reported in bold.

Environments. The experiments have been realised on the *deep-sea-treasure-concave-v0* (DST, Figure 3.1), a well-known benchmark environment available in MO-Gymnasium [Vamplew et al., 2011, Alegre et al., 2022]. The particularity of this environment is that it exposes a concave PF, making it impossible for decomposition-based algorithms using linear scalarization to discover the full PF (see Figure 2.9).

In this problem, the agent controls a submarine starting on the top left corner of the map. It perceives only its x, and y coordinates on the map. This environment has two objectives, the first one is the value of the treasures, and the second one is time. Whenever the agent performs an action, it obtains -1 on the time objective. If the agent encounters a treasure, it obtains the value of the treasure on its first objective and the episode is ended. An episode cannot last more than 1000 time steps.

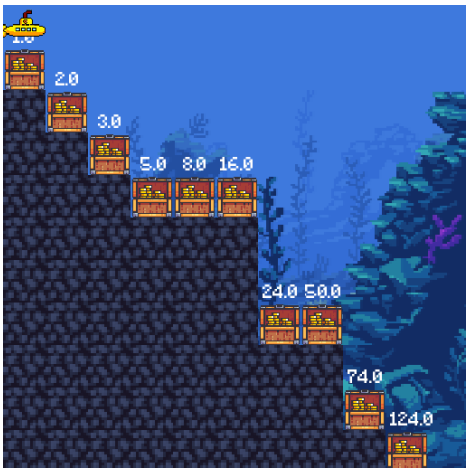


FIGURE 3.1: *deep-sea-treasure-concave-v0*

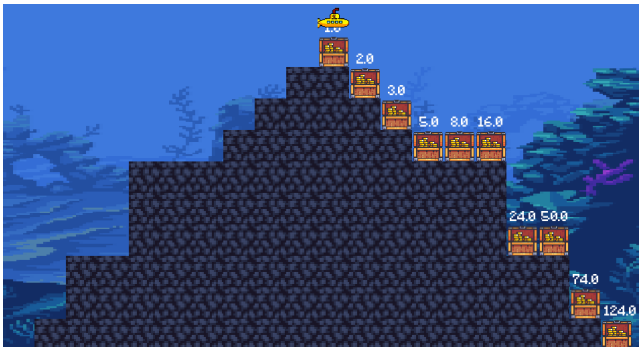


FIGURE 3.2: *deep-sea-treasure-mirrored-v0*

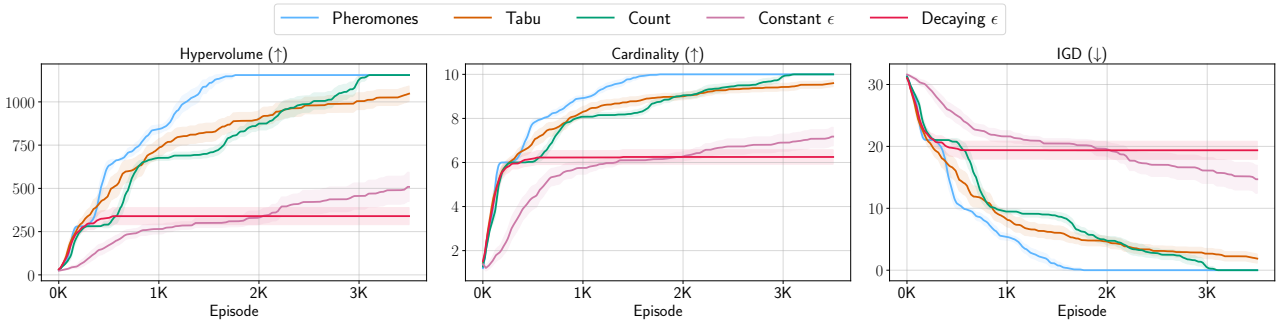


FIGURE 3.3: Results for 40 different random seeds on the DST.

Even though it is one of the most common MORL environments, the DST features a rather small state-action space, making the learning phase in this environment too easy when compared to real problems. Hence, to study the performance of our exploration strategies on a larger search space, we introduce a harder version of the DST: the *deep-sea-treasure-mirrored-v0* (MDST), which is presented in Figure 3.2. It is similar to the DST, except the map has been mirrored without treasures and the agent starts in the middle of the top row. This problem is thus harder to explore as it contains more state-action pairs, and the agent can spend a lot of time on the left part of the map which does not contain any treasure. Moreover, the narrow corridor between the parts of the map makes it challenging for an agent to recover from choosing the left side early. Finally, the agent can choose to go to the treasures on the top of the map easily whereas the treasures on the bottom require much more exploratory moves. The Pareto front is the same as in the DST.

Implementation details. The learning part of Algorithm 3.1 was implemented using PQL (Algorithm 2.4), with $\gamma = 1$. The hypervolume was used as a heuristic function. The code and results from the experiments can be found on https://github.com/ffelten/PMORL/tree/icaart. Table 3.1 summarizes the hyperparameter values for each strategy used in our experiments.

Metaheuristic	Hyperparameters
Tabu	$\tau = 150$
Count	$\alpha = 1$, $\beta = 3$, $m = 1$
Pheromones	$\alpha = 1$, $\beta = 2$, $\rho = 0.9$, $m = 1$
Constant ϵ	$\epsilon = 0.4$
Decaying ϵ	$\epsilon = 0.997^{episode}$

TABLE 3.1: Hyperparameter values of the various metaheuristics used in the experiments.

Eps. \ Metas	Pheromones	Tabu	Count	Constant ϵ	Decaying ϵ
500	**634.8** ± 92.8	**460.9** ± 205.1	290.3 ± 58.4	170.4 ± 105.1	330.4 ± 10.7
1000	**843.3** ± 76.1	**736.3** ± 166.3	**676.2** ± 46.8	264.8 ± 89.9	339.4 ± 144.1
1500	**1110** ± 107.1	**824.3** ± 167.3	**703.8** ± 76.3	300.1 ± 83.2	339.6 ± 156.6
2000	**1155** ± 0	**903.9** ± 152	**874.5** ± 117.5	333.1 ± 126.1	339.6 ± 157
2500	**1155** ± 0	**979.2** ± 153	**987.6** ± 151.8	420.9 ± 215.5	339.6 ± 157
3000	**1155** ± 0	**1004.1** ± 154.8	**1132.5** ± 79	455.7 ± 240.8	339.6 ± 157
3500	**1155** ± 0	**1047.6** ± 141.7	**1155** ± 0	508.7 ± 269.2	339.6 ± 157

TABLE 3.2: Hypervolume mean and standard deviation from the vectors in the starting state for different strategies every 500 episodes on the DST. Bold figures signify the means are statistically different from the means of Constant ϵ and Decaying ϵ according to ANOVA and Tukey's tests.

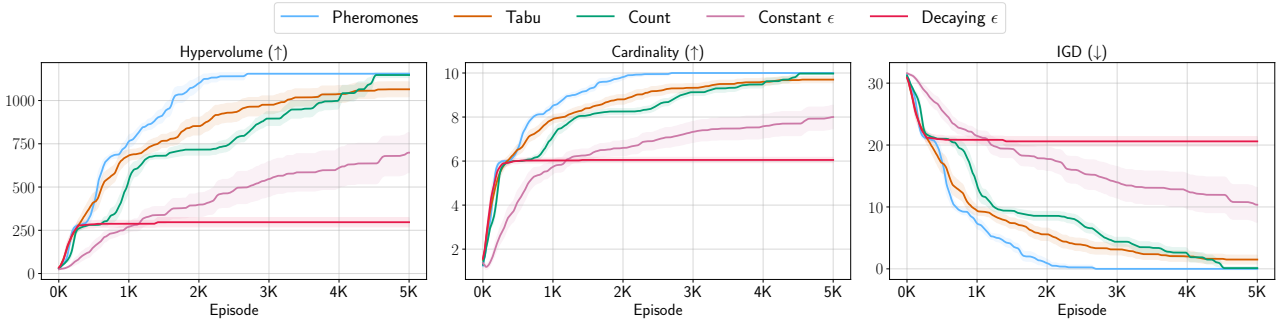


FIGURE 3.4: Results for 40 different random seeds on the Mirrored DST.

Deep-sea-treasure results. Figure 3.3 presents the learning curves based on performance indicators computed from the PF seen from the starting state ($\mathcal{F}(s_0)$) through the training phase. The reference point $\mathbf{z}_{\text{ref}}$ was set to $(0, -25)$ for both training and evaluation. In this environment, the three proposed exploration mechanisms (Pheromones, Tabu, Count) show better performance than the state-of-the-art approaches.

In particular, the pheromones-based metaheuristic can retrieve all the points in the front after ≈ 1750 episodes on average. The count-based exploration strategy requires more episodes to find all the points as it exploits multiple times close states before choosing to explore new further states. The tabu-based exploration strategy is not able to learn the full front in the given training window. This can probably be enhanced with better parameterization of the tabu list size.

Regarding the state-of-the-art exploration strategies, decaying ϵ shows a rapid convergence at the beginning of the training phase — when it mostly follows a random policy — but quickly stops finding new policies due to a lack of exploration. This was expected since it is starting to converge to a greedy policy. This confirms the convergence of the behavioral policy to a greedy policy is not necessary and can be harmful in the *a posteriori* setting. Constant ϵ is the worst strategy in the early episodes but shows better performance than the other state-of-the-art strategy in the end as exploration is maintained. It can also be noted that the confidence interval of this strategy grows with the number of episodes. This is due to the randomness of the algorithm, which the newly introduced strategies do not seem to suffer from. This could cause an unreliable performance of the agent in real applications as it probably would not be trained multiple times for each problem.

Table 3.2 presents the mean and standard deviation of the hypervolume indicator every 500 episodes for each of the studied strategies. Here also, the three proposed metaheuristics outperform the state-of-the-art methods. This is confirmed by the ANOVA and Tukey tests; the mean hypervolume of the proposed strategies are statistically different from the ones of the ϵ-greedy-based approaches in all cases except one, count-based after 500 episodes.

Eps. \ Metas	Pheromones	Tabu	Count	Constant ϵ	Decaying ϵ
500	**405.1** ± 159.1	**413.8** ± 169.9	281 ± 0	154.3 ± 102.9	284.1 ± 66.8
1000	**765.3** ± 112.7	**685.1** ± 96.8	**542.1** ± 179.7	270.4 ± 94.2	287.3 ± 63.4
1500	**937.5** ± 133.9	**767.1** ± 119.6	**680.8** ± 54.5	338.6 ± 154.8	296.7 ± 85.5
2000	**1102.5** ± 114	**852.2** ± 162.8	**716.1** ± 100	398.4 ± 217.5	296.7 ± 85.5
2500	**1140** ± 65.4	**933.5** ± 172.4	**771.3** ± 112.9	477.4 ± 262.5	296.7 ± 85.5
3000	**1155** ± 0	**975.4** ± 163.2	**895.5** ± 126.8	545.9 ± 320	296.7 ± 85.5
3500	**1155** ± 0	**1017.9** ± 157.9	**952.8** ± 151.6	584.8 ± 343.5	296.7 ± 85.5
4000	**1155** ± 0	**1037.7** ± 154.2	**997.5** ± 149.8	609.4 ± 365.2	296.7 ± 85.5
4500	**1155** ± 0	**1056.3** ± 142.4	**1140** ± 65.4	636 ± 372.6	296.7 ± 85.5
5000	**1155** ± 0	**1065** ± 137.5	**1147.5** ± 46.8	698.8 ± 368.5	296.7 ± 85.5

TABLE 3.3: Hypervolume mean and standard deviation from the vectors in the starting state for different strategies every 500 episodes on the MDST. Bold figures signify the means are statistically different from the means of $C\epsilon$ and Dϵ according to ANOVA and Tukey's tests.

Mirrored Deep-sea-treasure results. Figure 3.4 shows the results of the experiments on the newly introduced MDST environment. The reference point $\mathbf{z}_{\text{ref}}$ was set to $(0, -55)$ for the training and to $(0, -25)$ for the evaluation. As this environment is more complex to learn, more training episodes are presented.

Similarly to the DST, in this new environment, the proposed strategies allow for faster learning than the state-of-the-art ones. The pheromones-based strategy is the only one able to find all the optimal policies in the given training window. Tabu and count-based strategies are performing well when compared to the two ϵ-greedy strategies but fail to find the full PF.

In this scenario again, decaying ϵ stops finding new policies as it becomes greedier and lacks exploration. Finally, constant ϵ behaves the same as in the other environment, *i.e.*, it keeps learning new points with episodes, but with a high variance.

As for the DST, Table 3.3 presents the numerical results of the experiments. The same conclusion as for the DST holds; metaheuristics inspired approaches are statistically better than state-of-the-art approaches in this environment.

3.1.5 Summary

In this section, a modular framework for multi-policy Pareto-based MORL training has been presented. It allows to change some of its parts without the need to change the others. From that framework, the possibility of applying state-of-the-art optimization methods to control the exploration in MORL was identified.

Three new exploration strategies were presented to solve the exploration problem in this setting: pheromones-based, count-based, and tabu-based. These are inspired by existing work in metaheuristics. All three proposed strategies perform better than the current state-of-the-art ϵ-greedy-based methods on the studied environments. Also, it is shown that behavioral policies that do not converge to become greedy perform better than converging ones in the learning phase. Moreover, a new benchmark called the Mirrored Deep Sea Treasure has been proposed. It is a harder version of the well-known Deep Sea Treasure.

3.2 A Framework for MORL Based on Decomposition

In Section 2.3.4, we introduced multi-objective RL based on decomposition (MORL/D) and demonstrated the applicability of existing techniques from MOO/D and RL to enhance the sample efficiency of naive implementations (*e.g.*, Algorithm 2.5). In this section, we propose a comprehensive framework that facilitates

Algorithm 3.6 MORL/D high-level framework. Blue parts emphasize concepts coming from RL while black parts come from MOO/D. See Algorithm 2.3 for comparison with MOO/D.

Input: Stopping criterion *stop*, Population size n, Scalarization method g, Exchange trigger *exch*, Environment *env*, Update passes u.
Output: The approximated Pareto set stored in the external archive population $\mathcal{PS}$.

1: $\Pi, \mathcal{W}, \mathcal{Z} = Initialize(n)$
2: $\mathcal{PS} = ParetoPrune(\Pi, \mathcal{F}(\Pi))$
3: $\mathcal{N} = InitializeNeighborhood(\Pi, \mathcal{W})$
4: $\mathcal{B} = \emptyset$
5: **while** $\neg stop$ **do**
6: $\quad \pi, \mathbf{w}, \mathbf{z} = Select(\Pi, \mathcal{W}, \mathcal{Z})$
7: $\quad experiences = Sample(env, \pi, g, \mathbf{w}, \mathbf{z}, exch)$
8: $\quad \mathcal{B} = UpdateBuffer(\mathcal{B}, experiences)$
9: $\quad \Pi = ImprovePolicies(\Pi, g, \mathcal{W}, \mathcal{B}, u)$
10: $\quad \mathcal{PS} = ParetoPrune(\mathcal{PS} \cup \Pi, \mathcal{F}(\mathcal{PS}) \cup \mathcal{F}(\Pi))$
11: $\quad \mathcal{W}, \mathcal{Z} = Adapt(\mathcal{W}, \mathcal{Z}, \mathcal{F}(\mathcal{PS}))$
12: $\quad \mathcal{N} = UpdateNeighborhood(\Pi, \mathcal{N}, \mathcal{W})$
13: $\quad Cooperate(\mathcal{N})$
14: **end while**
15: **return** $\mathcal{PS}$

the seamless transfer of techniques from MOO and RL into the context of MORL/D. This framework is built upon the earlier analysis and taxonomy. By employing such a framework, existing techniques can be easily adapted, allowing practitioners to leverage advancements in both RL and MOO. It is worth emphasizing that even though many MORL works rely on decomposition. This is the first time a general modular framework making a clear connection between RL, MOO, and MORL is proposed.

This section initiates with a formal definition of the framework, proceeds to a discussion on making informed choices regarding instantiations, and ultimately presents results from various framework instantiations on diverse benchmark problems.

3.2.1 The MORL/D Framework

Algorithm 3.6 defines a high-level skeleton of the MORL/D framework, a middle-ground technique between MOO/D and RL aimed at finding a set of solutions to MOMDPs. Here, the population of policies (solutions) is denoted Π to clearly identify the different solution concepts with RL and MOO. The algorithm starts by initializing the policies, weights, reference points, Pareto archive, neighborhoods, and buffer (lines 1–4). At each iteration, the algorithm samples some experiences from the environment by following a chosen policy (lines 6–7). These experiences are then included in an experience buffer and used to improve the policies by sampling u batches from the buffer for each policy in the population (lines 8–9). After improvement, the policies are evaluated, and the Pareto optimal policies are included in the Pareto archive (line 10). Then, the weights, reference points, and neighborhood are adapted (lines 11–12), and subproblems can cooperate with each other (line 13). Finally, the set of non-dominated policies is returned (line 15).

As a reminder, the design choices exposed in the taxonomy (Figure 2.12) are listed below along with their associated variable and function names in Algorithm 3.6:

- **Scalarization:** The scalarization function employed to transform the MO problem into several single-objective problems. Represented by g;

- **Weight vectors:** The way the weight vectors used in the scalarization are generated or adapted based on the current state of the estimated PF. Represented by $\mathcal{W}, \mathbf{w}, \mathit{Adapt}, \mathit{Initialize}$;
- **Reference points:** Some scalarization functions require setting reference points, this addresses how they are generated and adapted. Represented by $\mathcal{Z}, \mathbf{z}, \mathit{Adapt}, \mathit{Initialize}$;
- **Cooperation:** How and when the subproblems share information with their neighbors to improve the sample efficiency of the overall algorithm. Also addresses how to determine subproblems that are part of the neighborhood. Represented by $\mathcal{N}$, $\mathit{InitializeNeighborhood}$, $\mathit{Cooperate}$, exch, $\mathit{UpdateNeighborhood}$;
- **Selection:** How, at each iteration, the policies, weight vector, and reference points are chosen to continue the learning process. Represented by Select;
- **Archive:** Whether a Pareto archive is used to store the Pareto optimal policies. Also addresses additional pruning criteria, such as crowding distance. Represented by $\mathcal{PS}, \mathcal{F}, \mathit{ParetoPrune}$;
- **Regression structure:** How the policies are encoded, can be tabular or function approximation. Also addresses questions of whether to perform multi-objective regression or not. Represented by Π;
- **Policy improvement:** The Bellman equation used to update the policies based on the gathered experiences. Represented by $\mathit{ImprovePolicies}$;
- **Buffer:** The criteria used for replacement and selection from the experience buffer. Represented by $\mathcal{B}$, $\mathit{UpdateBuffer}$;
- **Sampling strategy:** How the actions are chosen when gathering new experiences. Represented by Sample.

3.2.2 How to Instantiate MORL/D?

The presented framework provides enough modularity to be instantiated using different design choices presented in Section 2.3.4 and form new algorithms. Yet, as MORL/D presents a combinatorial number of potential instantiations, it may be hard to choose which technique to apply to solve a given problem. This section discusses the decisions of choosing which MORL/D variant to apply given a problem.

First, we provide a formal decision process regarding linear and non-linear scalarization choice in Section 2.3.4.3 and a graphical representation is illustrated in Figure 2.14. Additionally, some combinations of techniques do not make sense. For example, conditioned regression may not be combined with transfer learning, and shared layers are not compatible with tabular algorithms. Hence, using common sense on the presented techniques, algorithm designers may be able to restrict the number of possibilities.

However, for some design choices, it is not possible to give absolute directions since a technique that performs well for a given environment might perform poorly for another, as the "no free lunch" theorem states [Wolpert and Macready, 1997]. Yet, our framework may open ways to solve such an issue by allowing to automate the design of MORL/D algorithms. In such a context, an optimization algorithm is applied to search the space of possible instantiations of MORL/D and decide which algorithmic choices to make in order to maximize its performance for a given problem. In RL, these kinds of approaches are referred to as "AutoRL" [Parker-Holder et al., 2022, Eimer et al., 2023, Felten et al., 2023b]. Hence, we believe the presented framework could be useful to extend such work to form an AutoMORL solver. Nevertheless, the next section illustrates in practice how we tackle different benchmark problems without such an automated solver at our disposal.

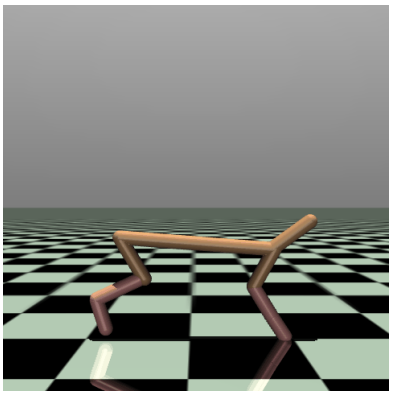


FIGURE 3.5: *mo-halfcheetah-v4*.

3.2.3 Experimental Settings

This section shows the results of different instantiations of the MORL/D framework. It has been instantiated to tackle two contrasting multi-objective benchmarks (*i.e.*, deep-sea-treasure and MO-Halfcheetah) [Vamplew et al., 2011, Alegre et al., 2022], showing how versatile the framework can be. Indeed, the two studied environments contain concave and convex PFs, and involve continuous and discrete observations and actions. To give a reference on the achieved performance, we compare MORL/D results against state-of-the-art methods.

Metrics. To assess the performance of our instantiations and compare with state-of-the-art methods, we use various performance indicators as presented in Section 2.2.3: IGD, hypervolume, expected utility, and sparsity. Each set of experiments has been run 10 times over various seeds for statistical robustness.

Environments. The first problem, *mo-halfcheetah-v4* (Figure 3.5), presents a multi-objective adaptation of the well-known Mujoco problems [Todorov et al., 2012]. In this scenario, the agent takes control of a bipedal robot and aims to mimic the running behavior of a cheetah. Similar to Example 1.1, the agent's objectives in this environment involve simultaneously maximizing its speed and minimizing its energy consumption. Unlike the original Mujoco implementation, which relies on a weighted sum with hard-coded weights to transform the problem into a single-objective MDP, the multi-objective version treats both objectives independently. Thus, it allows learning various trade-offs for each of these two objectives.

The second problem of interest, *deep-sea-treasure-concave-v0* (Figure 3.1), was presented earlier in Section 3.1. As a reminder, this benchmark problem holds particular significance due to the presence of a known PF that exhibits a concave shape. This concavity poses a challenge for MORL/D methods that rely on linear scalarization, as discussed in Section 2.3.4.3.

Implementation details. We reuse the data hosted by Open RL Benchmark (ORLB) [Huang et al., 2024] to plot metrics from state-of-the-art methods. MORL/D has been implemented using utilities from the MORL-Baselines [Felten et al., 2023a] (presented later in Section 4.2) and Pymoo projects [Blank and Deb, 2020]. Our experiments have been run on the high-performance computer of the University of Luxembourg [Varrette et al., 2014].[1] All values of hyperparameters set during our experiments are reported in Tables 3.4 and 3.5. The raw data of our training results are also available in ORLB.

[1]The code used for experiments is available in MORL-Baselines https://github.com/LucasAlegre/morl-baselines.

	Hyperparameter	Value
MORL/D	Population size	$n = 6$
	Episodes for policy evaluation	5
	Update passes	$u = 10$
	Total environments steps	$5e6$
	Exchange trigger	Every 50,000 steps
	Scalarization	Weighted sum
	Weight adaptation	PSA, $\delta = 1.1$
SAC	Buffer size	10^6
	Gamma	0.99
	Target smoothing coefficient	0.005
	Batch size	256
	Steps before learning	15,000
	Hidden neurons in Neural Networks	[256;256]
	Activation function in Neural Networks	ReLU
	Actor learning rate	$3e{-}4$
	Critic learning rate	10^{-3}
	Actor training frequency	2
	Target training frequency	1
	Entropy regularization coefficient	automatic

TABLE 3.4: Hyperparameters for MORL/D on *mo-halfcheetah-v4*.

	Hyperparameter	Value
MORL/D	Population size	$n = 10$
	Episodes for policy evaluation	5
	Update passes	$u = 10$
	Total environments steps	$4e5$
	Exchange trigger	Every 1,000 steps
	Scalarization	Chebyshev
	Optimization criterion	ESR
	Weight adaptation	PSA: $\delta = 1.1$
EUPG	Buffer size	10^5
	Gamma	0.99
	Hidden neurons in NN	[32;32]
	Activation function in NN	Tanh
	Learning rate	$1e{-}4$

TABLE 3.5: Hyperparameters for MORL/D on *deep-sea-treasure-concave-v0*.

3.2.4 Solving *mo-halfcheetah-v4*

This section explains how we tackled the *mo-halfcheetah-v4* problem using MORL/D and compares results against state-of-the-art methods implemented in MORL-Baselines [Felten et al., 2023a].

3.2.4.1 MORL/D Variants

To tackle this task, we initially performed an ablation study for some components of MORL/D (Figure 2.12). Subsequently, we compare the best version found against various existing MORL algorithms.

The first MORL/D instantiation, which we refer to as MORL/D vanilla, neither performs any cooperation nor weight vector adaptation. We then try to add cooperation and weight vector adaptation to this vanilla algorithm and examine the results. In practice, we tried adding PSA's method to adapt weights (Equation 2.11), and a shared buffer as cooperation mechanism. When relying on one technique, we suffix the technique acronym to MORL/D, *e.g.*, MORL/D SB PSA refers to a variant of the algorithm that implements a shared buffer and PSA's weight vector adaptation. It is worth noting that more sophisticated schemes coming from MOO or RL literature can easily be integrated too. Our instantiation is discussed in more detail below.

For this continuous problem, each policy in the population relies on a scalarized multi-objective version of the SAC algorithm [Haarnoja et al., 2018]. Practically, the SAC implementation from Huang et al. [2022b] was modified by including multi-objective critics and adding a scalarization function.

Scalarization. Because of its simplicity to implement, and to avoid making a choice between ESR and SER settings, the weighted sum has been chosen to instantiate MORL/D on this environment.

Weight vectors. For initialization, weight vectors are uniformly generated on a unit simplex using the Riesz s-Energy method from Blank et al. [2021]. Moreover, some MORL/D variants perform PSA's weight vector adaptation (Equation 2.11) every 50,000 steps.[2]

Cooperation. MORL/D vanilla does not implement any cooperation mechanism, whereas MORL/D SB implements shared buffer across the entire population. In the latter, the neighborhood is all other policies in the population and the exchange happens continuously since the buffer is shared by everyone.

Selection. In all variants, each policy is attached to given weights (which can be adapted) and trained using those. To uniformly train all policies, candidates are chosen in a roulette-wheel fashion to sample the experiences.

Archive. To ensure no performance loss after weight adaptation, a Pareto archive has been implemented using the Pareto dominance criterion as sole pruning function. The archive stores snapshots of the SAC policies leading to Pareto optimal points.

Regression structure. The studied MORL/D variants are actor-critic methods based on neural networks. The critics have been modified to implement a multi-objective regression, *i.e.*, each critic outputs m values.

Policy improvement. The scalarization function is applied on the muti-objective estimates from the critics to transform them into scalars. This allows falling back to the original implementation of the Bellman update with scalarized RL.

Buffer. Both MORL/D variants use experience buffers with a recency criterion for storage and sample uniformly from the buffers. In the case of MORL/D SB variants, a single buffer is shared among all the policies.

Sampling strategy. In all variations, samples are collected by following the selected candidate policy, *i.e.*, policy following. Note that policy updates are also performed on the currently followed policy while following the policy, as in the original implementation of SAC.

3.2.4.2 Experimental Results

This section first describes how we solved the halfcheetah problem, first by performing an ablation study on various MORL/D instantiations, then comparing the best MORL/D variant against state-of-the-art algorithms.

[2] In our experience, it is also beneficial to normalize the reward components to facilitate the finding of weight vectors which lead to a diverse PF when the scale between objective values is different.

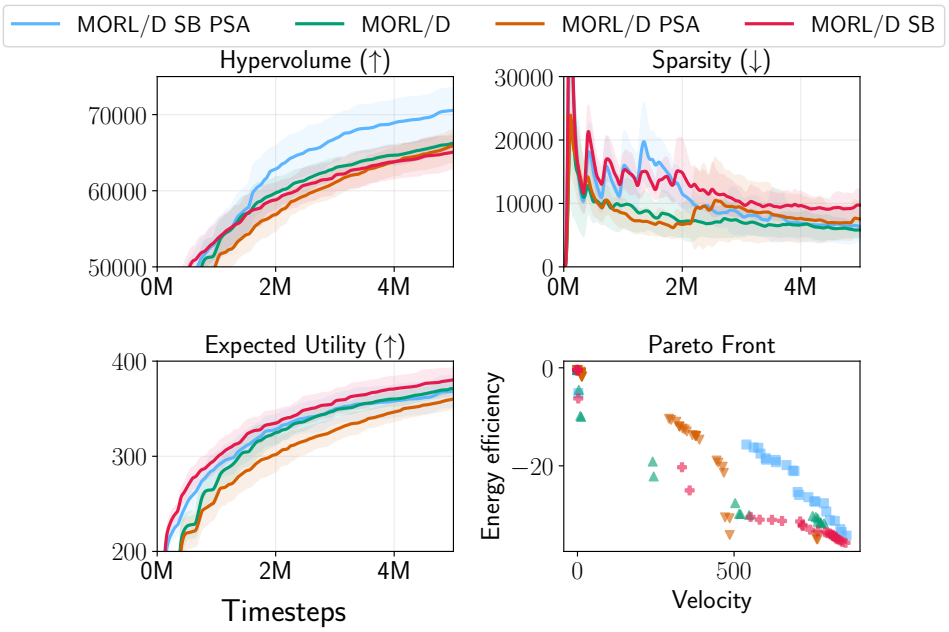


FIGURE 3.6: Average and 95% confidence interval of various metrics over timesteps on *mo-halfcheetah-v4* for variants of MORL/D. The rightmost plot is the resulting PF of each method's best run (best hypervolume).

Ablation study. Figure 3.6 shows the results of an ablation study of various versions of MORL/D on the *mo-halfcheetah* problem. A salient observation is that all MORL/D variants consistently improve their Pareto sets and PFs due to the utilization of the Pareto archive. When it comes to hypervolume (reference point $(-100, -100)$), it is evident that MORL/D with a shared buffer and PSA weight adaptation (MORL/D SB PSA) appears to outperform other variants in general. However, in terms of metrics such as sparsity and expected utility, no particular variant emerges as superior to the others.

The right plot of the PF illustrates how policies are distributed across the objective space for their best run.[3] Notably, this graph reveals an unequal scaling of objectives, which implies that scalarized values, based on uniformly spaced weights and metrics like expected utility, may exhibit bias in favor of objectives with larger scales. This phenomenon is corroborated by the fact that most of the Pareto optimal points are located on the right side of the plot, primarily because the velocity objective exhibits a larger scale compared to energy efficiency. Even though we normalize the rewards using a wrapper for this problem, it appears that this normalization is insufficient to learn a continuous PF. Nevertheless, the plot underscores that adding cooperation and weight adaptation to the vanilla MORL/D improves its performance.

MORL/D vs. state of the art. We now compare MORL/D SB PSA to the state-of-the-art methods in Figure 3.7 to gauge its performance against established baselines. Specifically, we evaluate our algorithm against prediction-guided MORL (PGMORL) [Xu et al., 2020a] and concave-augmented Pareto Q-Learning (CAPQL) [Lu et al., 2023]. First, PGMORL is an evolutionary algorithm that maintains a population of policies learned using a scalarized (weighted sum) version of PPO [Schulman et al., 2017]. It introduces a prediction model that aims to forecast improvements on the PF from previous policy evaluations and specific weight vectors. This predictive model is then used to choose pairs of policies and weight vectors that are expected to result in the most significant predicted improvements, following a tournament-style selection approach. Second, CAPQL closely resembles the instantiation of MORL/D we have employed to address this problem. CAPQL, indeed, relies on scalarized SAC using a weighted sum. Nevertheless, it distinguishes itself by relying on conditioned

[3]It is worth emphasizing that such a plot reflects the performance of only one run, while the metric plots reflect the general performance over multiple runs.

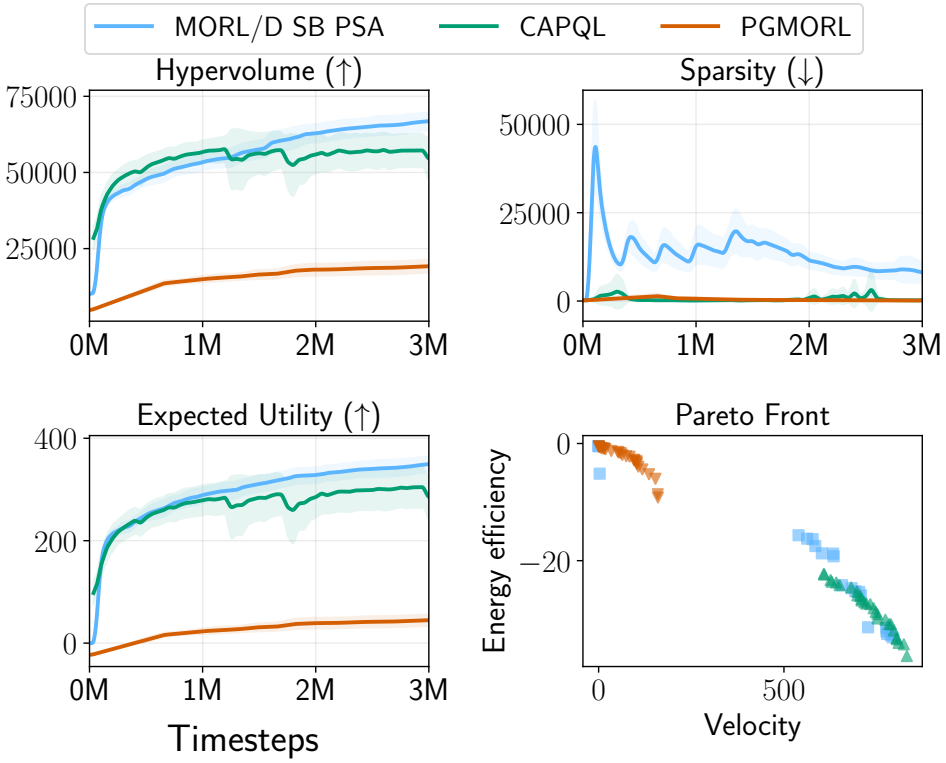


FIGURE 3.7: Average and 95% confidence interval of various metrics over timesteps on *mo-halfcheetah-v4*, MORL/D compared against state-of-the-art methods. The rightmost plot is the PF of each method's best run (best hypervolume).

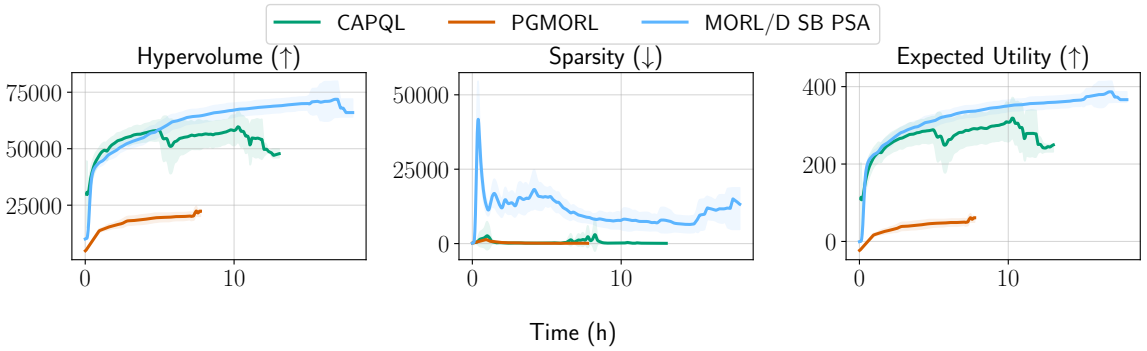


FIGURE 3.8: Comparisons in terms of runtime on *mo-halfcheetah-v4*.

regression and randomly sampling new weight vectors for each environment step. These methods have been chosen because of their ability to deal with environments displaying continuous state and action spaces.

Based on the data presented in Figure 3.7, our implementation achieves performance that is comparable to or even superior to the state of the art, particularly in terms of hypervolume and expected utility. This, in particular, contradicts the current belief that conditioned regression-based methods are more efficient than relying on a population of networks [Abels et al., 2019]. It is worth noting that the performance of CAPQL exhibits occasional drops during its training phase, which we suspect may be attributed to the conditioned neural network employed in the algorithm that forgets previously learned policies. This issue does not arise when using multiple neural networks in conjunction with a Pareto archive. For fairer comparisons, we also provide performance in terms of runtime in Figure 3.8. The same conclusions hold: MORL/D shows better performance than the baseline methods.

Additionally, the PF plot in Figure 3.7 reveals a noteworthy observation: while other algorithms tend to produce nearly continuous PFs on one side of the objective space, our algorithm discovers policies on both ends, contributing to improved diversity. However, sparsity appears to be more favorable in the case of the other algorithms. We believe that this highlights a limitation of this state-of-the-art metric: it assesses distance based on the points found by the algorithm rather than considering the entire objective space. Consequently, an algorithm that locates only a few closely clustered points may exhibit a low sparsity score despite providing smaller diversity.

3.2.5 Solving *deep-sea-treasure-concave-v0*

This section illustrates how MORL/D can be used to solve problems involving PFs with concave parts.

3.2.5.1 MORL/D Variant

In this section, our MORL/D algorithm depends on the expected utility policy gradient algorithm (EUPG) [Roijers et al., 2018]. EUPG is a single-policy ESR algorithm able to learn policies with non-linear scalarization. Employing such an algorithm with different weights enables us to capture policies within the concave part of the PF, which is uncommon in the current MORL literature. This section demonstrates how MORL/D can serve as a framework for converting pre-existing single-policy MORL algorithms into multi-policy ones.

Scalarization. In this case, we are interested in finding the concave points in the PF. Hence, we rely on the Chebyshev function (Section 2.2.5.1), which is a non-linear scalarization.

Reference points. The Chebyshev scalarization necessitates using a utopian reference point $\mathbf{z}$. In many cases, this reference point is hard to set in advance. Hence, we propose to automatically adapt it over the course of the learning process by setting $\mathbf{z}$ to be the maximum value observed for each objective, plus a factor $\tau = 0.5$.

Weight vectors. As for *mo-halfcheetah*, the Riesz s-Energy method is used to uniformly generate weight vectors. Moreover, PSA's weight vectors adaptation (Equation 2.11) is used every 1,000 steps.

Cooperation. No cooperation has been implemented on this problem.

Selection. Each policy is attached to given weights (which can be adapted) and trained using those. To uniformly train all policies, candidates are chosen in a roulette-wheel fashion to sample the next experiences.

Archive. A Pareto archive has been implemented using the Pareto dominance criterion as sole pruning function. The archive stores snapshots of the EUPG policies leading to Pareto optimal points.

Regression structure. EUPG relies on a single NN to model the policy. Interestingly, it proposes to condition the NN on the accrued reward to allow learning ESR policies with non-linear scalarization.

Policy improvement. This MORL/D variant relies directly on EUPG's policy improvement.

Buffer. This algorithm uses experience buffers with a recency criterion for storage and samples uniformly from the buffers.

Sampling strategy. Similar to the previous algorithm, samples are collected following the selected candidate policy, *i.e.*, policy following. Note that policy updates are also performed on the currently followed policy.

3.2.5.2 Experimental Results

Figure 3.9 displays the training outcomes of our MORL/D variant, which relies on EUPG as its underlying algorithm. Additionally, we present results for a comparative analysis involving multi-policy multi-objective Q-learning (MPMOQL) [Van Moffaert et al., 2013], a tabular algorithm that depends on multi-objective regression

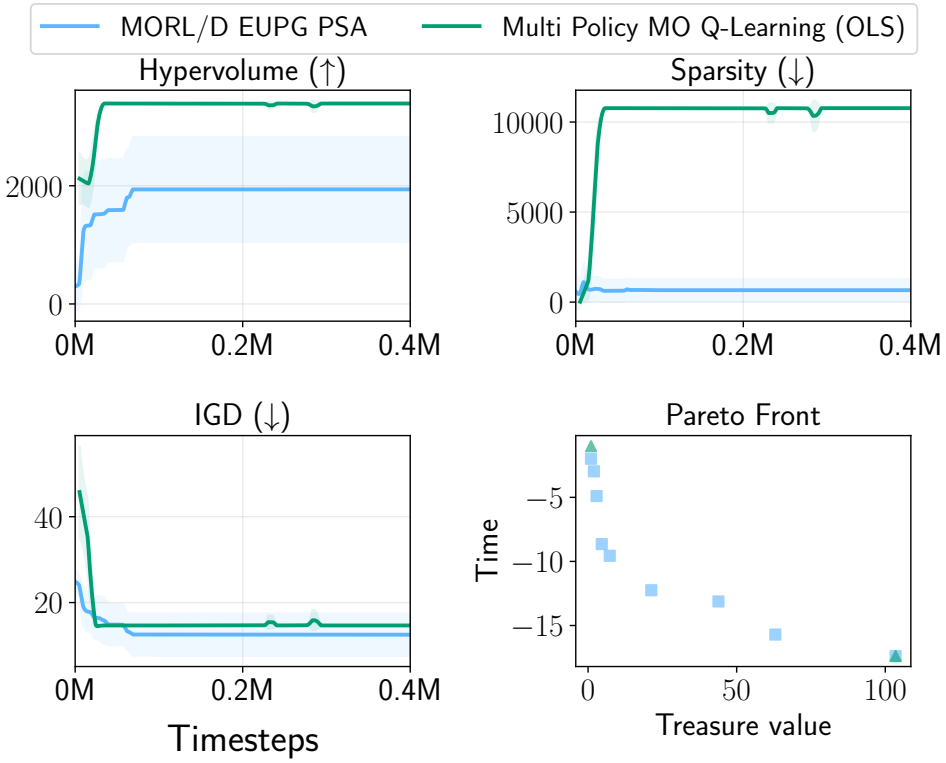


FIGURE 3.9: Average and 95% confidence interval of various metrics over timesteps on *deep-sea-treasure-concave-v0*. The rightmost plot is the resulting PF of each method's best run (best hypervolume).

and linear scalarization. It is essentially MPQL (Algorithm 2.5) with multi-objective Q-values. MPMOQL is run multiple times with various weights generated using optimistic linear support (OLS) [Roijers et al., 2015b] to learn various trade-offs. It is worth emphasizing again that the use of linear scalarization restricts this algorithm from effectively learning points within the concave part of the Pareto front. We used $(0, -50)$ as a reference point for the hypervolume computation. Note that we do not report expected utility in this case, since the metric supposes linear utility, which would not capture any contribution from the concave points in the PF.

Upon examining the plots, it appears that MORL/D performs less favorably in terms of hypervolume compared to MPMOQL. However, when assessing diversity and convergence metrics, a different picture emerges: MORL/D outperforms MPMOQL in terms of sparsity and IGD, respectively. Furthermore, the PF revealed by the best-performing runs clearly demonstrates that MORL/D can capture points within the concave portion of the PF, whereas MPMOQL with linear scalarization can only capture extreme points along the convex hull.

Upon closer examination, we saw that MPMOQL consistently captures the two extreme points, while MORL/D occasionally struggles to capture points on the right side of the Pareto front, primarily due to limited exploration in EUPG. Notably, the two extreme points captured by MPMOQL result in a very high hypervolume, overshadowing the contributions from the points in the concave region. Consequently, even if MORL/D manages to capture a few points in the concave region, its hypervolume remains lower than that of MPMOQL, which captures only the two extreme points. This underscores the limitation of relying solely on a single performance metric, such as hypervolume or sparsity in the previous example, for algorithm comparison. Indeed, attempting to condense such a wealth of information into a single scalar value comes with inherent trade-offs. Therefore, we advocate for the utilization of multiple metrics and the visualization of PFs whenever possible as a more comprehensive approach.

3.2.6 Summary

In this section, we applied the techniques outlined in MORL/D (Section 2.3.4) in practical scenarios. The approach involves employing a scalarization function to decompose multi-objective problems into individual single-objective problems. We introduced a unified framework grounded in the established taxonomy. This framework was adapted in various ways to formulate dedicated algorithms, and experiments were conducted across diverse benchmark problems.

These experiments effectively showcased the versatility of MORL/D in addressing a wide spectrum of challenges (*e.g.*, concave and convex PFs) through easily customizable adjustments in the instantiation of the framework. Notably, the results demonstrated how existing knowledge from MOO and RL could be seamlessly transferred to MORL, as exemplified by techniques like weight vector adaptation, leading to competitive performance compared to current state-of-the-art methods. Moreover, our experiments have shown that MORL/D techniques are able to leverage functional approximation to scale to higher-dimensional problems, such as continuous state and action spaces. Finally, the experimental outcomes and ensuing discussions brought to light crucial considerations, particularly emphasizing the limitations of relying solely on a single performance metric for algorithm evaluation.

3.3 Conclusion

In this chapter, we introduced two advancements that extended the current state of the art in MORL by incorporating established methodologies from MOO. Building upon the comprehensive analysis presented in Sections 2.1–2.3, we proposed the integration of MOO techniques into both Pareto-based MORL and decomposition-based MORL.

Our initial contribution introduced a flexible framework enabling rapid exploration strategy modifications within Pareto-based MORL. Through this framework, the utilization of metaheuristics demonstrated superior performance compared to conventional ϵ-greedy strategies. Subsequently, our second contribution outlined a versatile MORL/D framework based on the taxonomy introduced in Section 2.3.4, facilitating the seamless adoption of techniques from both RL and MOO/D into MORL/D to create new MORL algorithms. Leveraging this framework, various algorithms have been derived to tackle diverse problem scenarios, showcasing competitive performance relative to the current state of the art.

Beyond these algorithmic advancements, a significant portion of this thesis work focuses on enhancing the practicality of MORL. The forthcoming chapter presents software contributions and practical insights for effectively deploying MORL in real-world applications.

Published Work

The contributions described in Section 3.1 were published in the proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART):

- Florian Felten, Grégoire Danoy, El-Ghazali Talbi, and Pascal Bouvry. Metaheuristics-based Exploration Strategies for Multi-Objective Reinforcement Learning:. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*, pages 662–673. SCITEPRESS - Science and Technology Publications, 2022. ISBN 978-989-758-547-0. doi: 10.5220/0010989100003116

The MORL/D framework and results discussed in Section 3.2 were published in the Journal of Artificial Intelligence Research (JAIR):

- Florian Felten, El-Ghazali Talbi, and Grégoire Danoy. Multi-Objective Reinforcement Learning Based on Decomposition: A Taxonomy and Framework. *Journal of Artificial Intelligence Research*, 79:679–723, February 2024a. ISSN 1076-9757. doi: 10.1613/jair.1.15702. URL https://www.jair.org/index.php/jair/article/view/15702

Chapter 4

Reliable MORL

Contents

Research in RL algorithms [Sutton and Barto, 2018] has garnered considerable attention recently, largely due to its impressive success in tackling a variety of complex problems [Mnih et al., 2015, Silver et al., 2016]. This surge in interest has led to a significant rise in the number of papers published in the field annually. However, despite this rapid growth, the development of robust and well-considered tools for proper evaluation practices has not kept pace, resulting in a reproducibility crisis within the RL community. Consequently, researchers sometimes question the reliability and reproducibility of results presented in influential papers [Agarwal et al., 2021, Patterson et al., 2023]. This reproducibility issue also hinders the deployment of RL solutions in real-world scenarios. Inheriting from these, MORL also faces similar issues. Several factors contribute to these in (MO)RL:

1. Lack of standardization in environment implementations: The absence of standardized environment implementations makes it difficult for researchers to replicate results presented in papers;

2. Insufficient paper details: Papers often lack crucial information, such as implementation optimizations, hyperparameter values, code versions, or the usage of wrappers [Brockman et al., 2016], essential for reproducibility and ensuring good performance of algorithms;

3. Time and cost investment in RL agent training: The significant time, cost, and energy required to train RL agents hampers researchers' ability to gather enough data for rigorous statistical analysis of results;

4. Inadequate tuning: Papers introducing new algorithms often compare them against inadequately tuned baselines when applied to new environments, resulting in unfair comparisons. Moreover, using default hyperparameters on new environments may result in poor performance of existing algorithms, limiting the application of RL in real-world scenarios.

Numerous studies have highlighted these challenges in RL. For instance, Engstrom et al. [2020] demonstrate that optimizations made at the implementation level of deep RL algorithms can yield more significant impacts than merely experimenting with different algorithms. Similarly, Huang et al. [2022a] have delved into the investigation of 37 specific code-level optimizations essential for attaining state-of-the-art results when deploying the PPO algorithm [Schulman et al., 2017]. Agarwal et al. [2021], Patterson et al. [2023] draw attention to the absence of statistically significant results in numerous publications within the field, despite many claiming to introduce techniques that surpass the current state of the art. Furthermore, it is widely recognized that the effectiveness of RL (and MORL) agents is highly dependent on the values of their hyperparameters [Henderson et al., 2018, Andrychowicz et al., 2021, Patterson et al., 2023]. This implies that in a new environment, an existing algorithm may exhibit poor performance when run with its default hyperparameters. Finally, baseline comparisons frequently involve contrasting new algorithms with state-of-the-art techniques. In this regard, Patterson et al. [2023] pinpoint the problem of unfairness when untuned baseline algorithms are pitted against finely tuned new algorithms.

To address concerns regarding reproducibility and reliability in single-objective RL, several libraries have emerged to provide more dependable baselines. One notable example is Gymnasium [Towers et al., 2023] (formerly OpenAI Gym [Brockman et al., 2016]), which provides a standardized API and a suite of reference environments for RL research and experimentation. Additionally, libraries like Stable-Baselines 3 [Raffin et al., 2021] and cleanRL [Huang et al., 2022b] provide well-tested, reliable, and consistently maintained implementations of RL algorithms. Recent initiatives such as Open RL Benchmark [Huang et al., 2024] streamline the analysis of learning metrics by making public a dataset of training results of algorithms on specific environments, which is fed by the RL community. These endeavors also address the reproducibility challenge by utilizing experiment tracking software like Weights and Biases [Biewald, 2020], facilitating the transparent sharing of hyperparameters and code version for different baselines.

As discussed by Cassimon et al. [2021], Hayes et al. [2022], various benchmark problems have been proposed to evaluate MORL methods. However, these benchmarks have not yet been made available via standardized APIs or centralized repositories. Arguably, this has made the experimental reproducibility of MORL algorithms harder, time-consuming, and error-prone. MORL-Glue [Vamplew et al., 2017c] represents an attempt to establish a centralized repository of MORL benchmarks. However, this library has not been widely adopted because it is implemented in Java and targets tabular problems, whilst the community currently focuses on using Python and deep RL techniques. Although some published works on MORL make their code publicly available [Yang et al., 2019, Abels et al., 2019, Xu et al., 2020a], these implementations are not often maintained by their authors. This makes reproducing empirical results in MORL a usually time-consuming and error-prone process.

To address these challenges, accelerate research, and ease the application of multi-objective RL, this chapter contributes a set of software libraries and studies that include:

1. MO-Gymnasium (Section 4.1, solving issue 1): An intuitive and flexible API for easily constructing new MORL environments, featuring over 20 pre-existing environments. This facilitates the seamless evaluation of algorithms across various domains as well as the introduction of new environments that are directly compatible with existing algorithm implementations.
2. MORL-Baselines (Section 4.2, solving issue 2): A collection of reliable and efficient implementations of state-of-the-art MORL algorithms, ensuring a solid foundation for advancing research. Importantly, all algorithms are compatible with MO-Gymnasium.
3. Open results (Section 4.3, solving issues 2 and 3): A thorough and robust set of benchmark results and comparisons of MORL-Baselines algorithms, tested across various challenging MO-Gymnasium environments.
4. Hyperparameter Optimization for MORL (Section 4.4, solving issue 4): A study focused on the challenges of setting hyperparameter values for MORL algorithms to ensure their fair comparisons and easier application in real-world scenarios.

4.1 MO-Gymnasium

In this section, we introduce MO-Gymnasium, an easy-to-use and flexible API designed to facilitate the rapid construction of novel MORL environments. Additionally, it provides a collection of regularly maintained and thoroughly tested environment implementations. The documentation for MO-Gymnasium is available at `https://mo-gymnasium.farama.org`. MO-Gymnasium aims at solving issue 1, providing reliable implementations of MORL environments, enumerated above.

Unlike traditional machine learning settings that often rely on fixed datasets, RL problems typically do not, making replication of experimental results challenging. Indeed, although MDP definitions are typically well-specified in research papers, their actual instantiation can be influenced by implementation decisions such as floating-point number rounding. Notably, even minor discrepancies in environment specifications can have a substantial impact on algorithm performance.

To mitigate these issues and enhance research in standard RL settings, Gymnasium [Towers et al., 2023][1] introduced an API and versioned environment collection, which has become the *de facto* standard library in RL with millions of downloads. However, Gymnasium is tailored for single-objective MDPs and lacks support for MORL domains. Even though it has been extended in various ways, such as PettingZoo [Terry et al., 2021] for MARL and D4MORL [Zhu et al., 2022] for offline MORL, there is currently no widely adopted library providing reliable MORL implementations.

To address this gap, this section presents MO-Gymnasium (previously MO-Gym [Alegre et al., 2022]), an extension designed explicitly for MORL research. MO-Gymnasium's API closely resembles Gymnasium's, inheriting its features and extending only where necessary. This ensures backward compatibility with a wide range of RL utilities provided by the Gymnasium ecosystem. The key distinction is that MO-Gymnasium returns reward vectors rather than scalars after action execution. MO-Gymnasium is available on PyPI and can be installed via `pip install mo-gymnasium`. Crucially, MO-Gymnasium is officially maintained by the Farama Foundation and recognized as a mature and well-supported library within the research community.

The rest of this section first presents the API of MO-Gymnasium, illustrates some notable environments, wrappers, and utilities provided by MO-Gymnasium, and finally concludes this part.

[1] Formerly known as OpenAI Gym [Brockman et al., 2016].

4.1.1 MO-Gymnasium API

```
1 import mo_gymnasium as mo_gym
2 env = mo_gym.make("deep-sea-treasure-concave-v0", render_mode="human")
3 state, info = env.reset(seed=42)
4 optimal_pf = env.pareto_front(gamma=0.98)
5 done = False
6 while not done:
7     action = my_agent.policy(state)
8     state, vector_reward, terminated, truncated, info = env.step(action)
9     done = terminated or truncated
```

Listing 1: MO-Gymnasium API usage.

Listing 1 illustrates the typical utilization of the MO-Gymnasium API, allowing for the definition of all aspects of the agent-MOMDP interaction depicted in Figure 2.10. Notably, this API involves implementing only a few well-defined functions to create a new environment.

The initial step entails instantiating an environment via its unique identifier along with a render mode option to determine the display type (interactive window, text in a terminal, or none) for the environment (line 2). Subsequently, the environment is reset using a specified random seed, facilitating control over stochastic transitions (line 3). Optionally, some environments provide the ability to extract an optimal PF from their definition (line 4).

During an episode, as long as the environment does not terminate, the agent takes action based on the current environment state (line 7). Following action execution, the environment provides the subsequent state, a reward vector, and signals indicating whether the episode is complete (line 8). The `terminated` signal notifies episode terminations, while `truncated` facilitates early environment resets, aiding agent learning by preventing prolonged episodes with no progress. The loop concludes upon reaching either termination or truncation.

The following paragraphs outline the functions necessary for defining a new environment in MO-Gymnasium.

Environment initialization. Each environment must include a constructor function. Typically, this constructor initializes environment-related variables and defines the `action_space`, `observation_space`, and `reward_space` of the agent. These specifications enable the extraction of dimensions and boundaries of action, state, and reward components on the learning algorithm side. These are typically used for initializing the size of the input and output layers of DNNs.

`reset` function. This function resets all environment variables, such as the current state, to their initial values. It optionally accepts a random seed as input, providing control over the random number generator used to manage stochastic transitions and ensure reproducibility. The function samples and returns the initial state ($s_0 \sim \mu_0$) of the environment along with an info dictionary.

`step` function. This function takes an action as an argument and updates the environment state (see Figure 2.10). After executing the action, the function returns the next state, reward vector, termination, and truncation signals, and a dictionary containing additional information. Importantly, this is the only function breaking the Gymnasium API since the reward vectors returned are NumPy arrays [Harris et al., 2020].

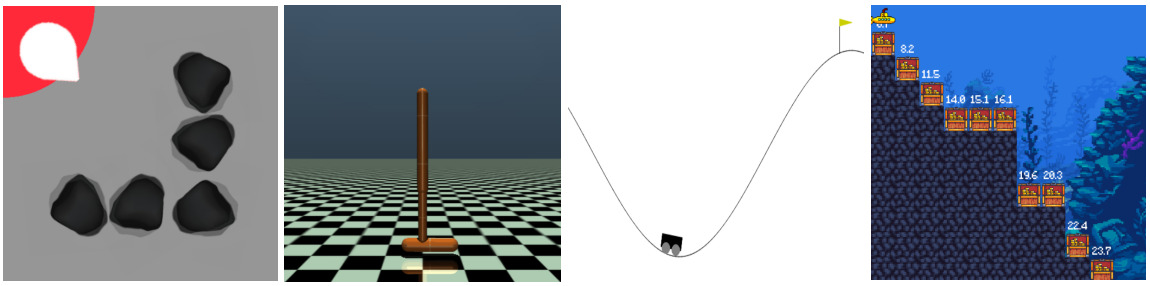


FIGURE 4.1: Visualization of some environments available in MO-Gymnasium.

`render` function. This function is responsible for rendering the environment and needs to be defined by the environment designer, as each environment may have its own rendering requirements.

`pareto_front` function. This function accepts a `gamma` parameter specifying the discount factor of the MOMDP. With this parameter and initialized variables, certain environments can compute their optimal PF. This capability allows the usage of performance metrics such as IGD, as discussed in Section 2.2.3, for environments with known optimal solutions.

4.1.2 Available Environments

At the time of writing, MO-Gymnasium includes a diverse selection of over 20 environments commonly featured in MORL literature. These environments encompass both discrete and continuous state and action spaces and include notable examples like *minecart* [Abels et al., 2019], *mo-hopper* [Xu et al., 2020a], and DST [Vamplew et al., 2011, Abels et al., 2019] (see Figure 4.1). Some of these environments are directly adapted from the single-objective RL literature. For instance, *mo-mountain-car-continuous* involves making a trade-off between quickly reaching the flag and minimizing fuel consumption, while *mo-halfcheetah* (Figure 3.5) involves balancing velocity and energy consumption. A complete list of supported environments is available on the documentation website: `https://mo-gymnasium.farama.org/`.

This large collection of environments provides designers with ample opportunities to thoroughly evaluate the performance of novel algorithms across various scenarios with little code change, as all environments respect the same API. Moreover, for reproducibility purposes, each environment is tagged with a version number, such as "*-v0*". Whenever an environment undergoes modifications that may impact algorithm performance, the version number of the domain is incremented accordingly.

4.1.3 Wrappers and Utilities

A key element of Gymnasium is the use *wrappers*, which may be used to *wrap* an environment and allow designers to modify one aspect of its dynamics, such as normalizing the observations returned to the agent. In MO-Gymnasium, specific wrappers tailored for MORL are introduced. These typically deal with the vectorial reward. These wrappers for example include *MONormalizeReward*, which normalizes a specified component of the reward vector, and *LinearReward*, a wrapper that linearly scalarizes the reward function of a MOMDP environment, effectively transforming it into a single-objective MDP. This latter feature provides compatibility of MO-Gymnasium with widely-used RL libraries like Stable-Baselines 3 and cleanRL, which are engineered to seamlessly interface with Gymnasium environments. It is worth noting that most of the wrappers in Gymnasium

that do not directly affect rewards, such as *NormalizeObservation* or *ClipAction*, can be readily applied to MO-Gymnasium environments without modification.

4.1.4 Summary

In this section, we introduced MO-Gymnasium, a standard API, a collection of MORL environments, and utilities for MORL research. Using this library, MORL researchers can easily validate their algorithmic contributions on varied domains. Moreover, non-MORL experts are able to provide new challenging environments by only implementing a few functions respecting the API. Then, state-of-the-art MORL algorithms, compatible with the MO-Gymnasium API can be used to solve the proposed MOMDPs. The contribution discussed in the next section provides such algorithms' implementations.

4.2 MORL-Baselines

In this section, we introduce MORL-Baselines, a comprehensive collection of reliable and efficient implementations of state-of-the-art MORL algorithms. This repository is designed to provide a solid foundation for advancing research in MORL and aims to simplify the application of MORL for individuals who may not be experts in the field. It is worth noting that MORL-Baselines is the first open-source repository to encompass a variety of MORL algorithms. Importantly, all algorithms included in MORL-Baselines are inherently compatible with the API provided by MO-Gymnasium. This means that end users can define their environments using the MO-Gymnasium API (as discussed in the previous section) and choose an algorithm from MORL-Baselines to find solutions to their MOMDP. This part aims to provide a solution to issue 2, providing reliable implementations of MORL algorithms, enumerated above.

As previously discussed, the performance of learning algorithms is intricately tied to their specific implementations [Engstrom et al., 2020]. Unfortunately, research papers often lack detailed discussions of implementation-specific optimizations, which can make reproducibility and comparisons challenging. To address this issue, some authors make their codebases accessible. However, these codebases are often not actively maintained and may become outdated, hindering replication efforts. Consequently, libraries like Stable-Baselines 3 [Raffin et al., 2021] and cleanRL [Huang et al., 2022b] have emerged with the goal of regularly maintaining state-of-the-art algorithms. These codebases are characterized by thorough documentation, extensive testing, and state-of-the-art performance on the algorithms they support. This enables researchers to begin developing novel methods by building upon existing implementations rather than starting from scratch. Additionally, these libraries offer valuable research tools such as efficient replay buffer implementations, performance reports, and evaluation methods. Importantly, these libraries empower individuals who may not be domain experts to utilize reliable implementations of RL algorithms for their specific problems. However, these libraries are limited to single-objective RL.

To overcome the lack of reliable implementations of MORL algorithms, we introduce MORL-Baselines. This library contains over 10 state-of-the-art MORL algorithms, all of which seamlessly integrate with MO-Gymnasium. MORL-Baselines provides researchers with a variety of features to assist in algorithm design, including methods for computing and analyzing Pareto fronts, evaluation with respect to various metrics, replay buffers, and experiment tracking tools. Finally, a detailed and comprehensive documentation website is available at `https://lucasalegre.github.io/morl-baselines`. Below, we list the algorithms currently available via MORL-Baselines, along with descriptions of the settings in which they are applicable. Then, some of the utilities provided by the library are also described.

Algorithm	Single or multi-policy	Utitlity function	Observation space	Action space
MOQL [Van Moffaert et al., 2013]	Single	Linear	Disc.	Disc.
EUPG [Roijers et al., 2018]	Single	Non-linear, ESR	Disc.	Disc.
MPMOQL [Van Moffaert et al., 2013]	Multi	Linear	Disc.	Disc.
PQL [Van Moffaert and Nowé, 2014] (Algorithm 2.4)	Multi	Non-linear, SER (*)	Disc.	Disc.
OLS [Roijers and Whiteson, 2017]	Multi	Linear	/ (**)	/ (**)
Envelope [Yang et al., 2019]	Multi	Linear	Cont.	Disc.
PGMORL [Xu et al., 2020a]	Multi	Linear	Cont.	Cont.
PCN [Reymond et al., 2022]	Multi	Non-linear, ESR/SER (*)	Cont.	Any
GPI-LS & GPI-PD [Alegre et al., 2023]	Multi	Linear	Cont.	Any
CAPQL [Lu et al., 2023]	Multi	Linear	Cont.	Cont.
MORL/D [Felten et al., 2024a] (Section 3.2)	Multi	Any	Any	Any

TABLE 4.1: Algorithms currently implemented in MORL-Baselines. (*) PCN and PQL are designed to tackle deterministic environments. (**) OLS is an algorithm-agnostic method for generating reward weights, or preferences; it does not assume any particular type of observation or action spaces.

4.2.1 Implemented Algorithms

Table 4.1 presents the algorithms currently supported by MORL-Baselines along with the MORL settings they address. Algorithms that utilize neural networks as function approximations for their regression structure were implemented using PyTorch [Paszke et al., 2019], while tabular algorithms rely on NumPy [Harris et al., 2020]. The description of algorithms in Table 4.1 distinguishes between those that yield a single policy (based on user-provided utility functions *a priori*) and those producing multiple policies (to approximate a PF). Furthermore, it is worth noting that certain algorithms optimize with regard to the ESR criterion, while others optimize regarding the SER criterion as indicated by the utility function column (see Section 2.3 and Figure 2.14). Notably, all multi-policy algorithms except PQL and PCN are decomposition-based algorithms, this reflects how central this technique is. Lastly, it is important to note that the algorithms within MORL-Baselines may accommodate various observation and action spaces, such as images.

Tabular algorithms. These algorithms use arrays or dictionaries for their regression structure. While these allow for keeping optimality guarantees of algorithms, they are usually unable to scale to large problems. The implemented tabular algorithms are listed below.

- Multi-objective Q-learning (MOQL) [Van Moffaert et al., 2013] is an extension of the classic tabular Q-learning algorithm [Watkins, 1989] that learns and stores multi-objective Q-values (multi-objective

regression, Section 2.3.4.2). A scalarization function is then used to convert these Q-values into a scalar quantity, allowing agents to select an action.

- Multi-policy MOQL (MPMOQL) has been presented in Section 3.2. It consists of running MOQL multiple times with different preferences (outer-loop style). It can be instantiated using different methods to generate weight vectors. Currently, MORL-Baselines supports randomly generated weight vectors, uniformly generated weight vectors, Optimistic Linear Support (OLS) [Roijers and Whiteson, 2017], and Generalized Policy Improvement Linear Support (GPI-LS) [Alegre et al., 2023].
- Pareto Q-learning (PQL) [Van Moffaert and Nowé, 2014] has been described in Section 2.3.3 and extended in Section 3.1. It aims at simultaneously learning all policies in the Pareto front by storing sets of non-dominated Q-values (Q-sets). These sets are then converted into scalars using multi-objective performance indicators or ranking functions (Sections 2.2.3–2.2.4, and 3.1.2) that guide the selection of actions during the learning phase. This algorithm is only compatible with deterministic environments.

Deep MORL algorithms. These algorithms use deep neural networks as function approximations in their regression structure. They are able to scale to problems with much greater dimensions. The deep MORL currently included in MORL-Baselines are listed below.

- The Expected Utility Policy Gradient (EUPG) algorithm [Roijers et al., 2018] has been utilized in Section 3.2. It introduces a policy gradient update capable of taking into account both the return achieved up to the current moment, as well as future returns, in order to optimize a policy in an ESR setting.
- The Envelope algorithm [Yang et al., 2019] uses conditioned regression (Section 2.3.4.3) and multi-objective regression (Section 2.3.4.2) to approximate the PF using a single DNN. Moreover, it relies on a multi-objective version of the Bellman update to optimize the DNN parameters to generalize over multiple weight vectors, see Equation 2.16.
- Prediction-Guided MORL (PGMORL) [Xu et al., 2020a] has been presented in Section 3.2. This algorithm is close to the MORL/D framework presented earlier: it is an evolutionary algorithm that maintains a population of policies learned using scalarized PPO [Schulman et al., 2017]. This algorithm focuses on predicting, at each iteration, the most promising weight vectors and policies to select for further training in order to more effectively improve the PF approximation. Moreover, a Pareto archive is maintained to keep the non-dominated candidates encountered over the course of the learning process.
- Pareto Conditioned Networks (PCN) [Reymond et al., 2022] employs a neural network conditioned on a given desired return per objective. This network is trained via supervised learning to predict which actions produce the desired return in deterministic environments.
- GPI-LS [Alegre et al., 2023] employs generalized policy improvement (GPI) [Barreto et al., 2017] to combine policies in its learned PF and select the weight vectors on which agents should train at each moment. GPI-Prioritized Dyna (GPI-PD) is a model-based extension of GPI-LS that uses a learned model of the environment and GPI to prioritize experiences in the replay buffer.
- Concave-Augmented Pareto Q-learning (CAPQL) [Lu et al., 2023] has also been presented in Section 3.2. It employs a multi-objective extension of SAC [Haarnoja et al., 2018] by conditioning the actor and critic networks on a weight vector. To learn a PF, it proposes to randomly change the weight vector used at each environment step.

- MORL/D is the implementation of the framework presented in Section 3.2. Notably, it enables learning PF approximations in all settings, as it abstracts the underlying RL procedure and focuses on the multi-objective parts. Technically, it could also be used in a tabular setting.

4.2.2 Utilities

Aside from the algorithms, MORL-Baselines provides a set of utilities allowing algorithm designers to construct novel MORL algorithms, deriving some of these from multi-objective optimization libraries like Pymoo [Blank and Deb, 2020]. Some of these utilities directly align with the taxonomy exposed in Section 2.3.4. All the utilities included in MORL-Baselines are listed below.

Pareto frontier filters. The library provides a set of functions that permit filtering Pareto-dominated points out of a set of points, following Equation 2.14.

Policies evaluation. Evaluating the values of policies currently undergoing training in the environment is essential for constructing a Pareto set of policies. MORL-Baselines provides specialized functions to standardize this evaluation process, ensuring fair comparisons. Additionally, the library includes a function that records all multi-objective metrics discussed in Section 2.2.3 to Weights and Biases (W&B) for consistent metric logging across multi-policy algorithms. Finally, it offers a method to initialize all components with a seed, including making PyTorch deterministic.

Performance indicators. MORL-Baselines implements all performance indicators defined in Section 2.2.3, including hypervolume, inverted generational distance, maximum utility loss, sparsity, cardinality, and expected utility. These functions accept an approximate Pareto front as input and produce scalar values representing the corresponding metric values. Some of these indicators, *e.g.*, IGD, and MUL, also require a reference Pareto front.

Scalarization functions. While most MORL algorithms rely on weighted sum scalarization, MORL-Baselines provides alternative scalarization schemes such as the Chebyshev scalarization function to capture optimal points in non-linear cases.

Weight vectors utils. MORL-Baselines supports various methods for generating weight vectors to scalarize vectorized rewards. These methods include random sampling, uniform generation in the objective simplex (*e.g.*, via the Riesz s-Energy method [Blank et al., 2021]), or adaptation based on the current PF approximation using OLS [Roijers and Whiteson, 2017] and GPI-LS [Alegre et al., 2023] methods.

Experience replay buffers. Experience replay buffers play a crucial role in RL algorithms by storing experience tuples collected during interaction with the environment. There are various approaches to designing these buffers, with particular attention required in the MORL case. MORL-Baselines offers implementations of these buffers, including a diversity-maintaining buffer proposed by Abels et al. [2019] and a prioritized experience replay buffer utilized by Alegre et al. [2023] (refer to Section 2.3.4 for additional background).

Neural networks. Neural networks serve as the main component in modern deep MORL algorithms. MORL-Baselines supports various architectures and enhancements related to the usage of neural networks, such as Convolutional Neural Networks [LeCun et al., 1998] for environments with image-based observations and Polyak updates [Lillicrap et al., 2016] for algorithms relying on target networks.

Miscellaneous utils. MORL-Baselines includes additional methods commonly used in MORL research, such as utilities for capturing GIFs from policy executions and implementing linearly decaying epsilon-greedy exploration as described by Mnih et al. [2015].

4.2.3 Summary

This section introduced MORL-Baselines, a collection of reliable implementations of MORL algorithms compatible with the MO-Gymnasium API. The library also provides implementations of utilities that are often used in MORL, such as a Pareto archive. Notably, some of these utilities align with the taxonomy presented in Section 2.3.4 and Figure 2.12. Based on the contributions exposed in the last two sections, it is possible to perform an empirical study of MORL algorithms from MORL-Baselines on various environments contained in MO-Gymnasium. The next section gives a proof-of-concept of such a study.

4.3 Publicly Available Benchmark Results

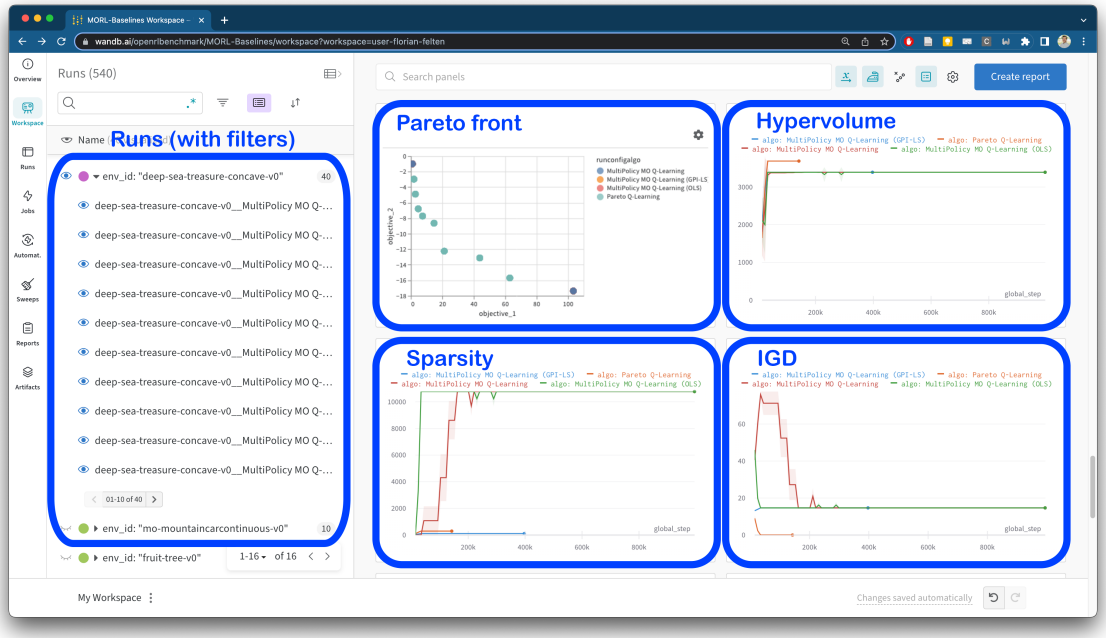


FIGURE 4.2: Overview of performance indicator values over the training phase in our dashboards.

In this section, we present a comprehensive array of benchmark results that assess the performance of MORL-Baselines algorithms across a diverse range of MO-Gymnasium environments.[2] These results are compiled into a dataset hosted within the Open RL Benchmark (ORLB) initiative [Huang et al., 2024], facilitating access and integration. The availability of such a dataset enables users conducting new experiments to seamlessly combine,

[2]These results are available for viewing and analysis at https://wandb.ai/openrlbenchmark/MORL-Baselines.

compare, and aggregate their new results with those already existing in the dataset. Furthermore, users can visualize these results in real-time using W&B dashboards. This access also permits users to manipulate raw data for subsequent analyses, compare performance metrics of new algorithms with state-of-the-art techniques without necessitating the retraining of the latter, and store associated hyperparameters, code version (git tag), and command lines for each run. For example, users can retrieve the approximate Pareto fronts identified by each algorithm within our benchmark through the W&B dashboard, accessible under the panel named "*eval/front*". Moreover, ORLB provides a command-line interface (CLI) allowing to extract hosted data and presenting them in plots for paper publication. Notably, all the plots reporting results in this thesis, except those in Section 3.1, have been generated using ORLB and its CLI. This contribution addresses issues 2 (storing hyperparameter values and code version) and 3 (costly training) as enumerated above.

4.3.1 Proof-of-concept Experiments using MORL-Baselines and MO-Gymnasium

In this section, MORL-Baselines and MO-Gymnasium are used to conduct a series of proof-of-concept empirical experiments and discuss their corresponding outcomes. The main goal of this section is to demonstrate the types of experiments, comparisons, and analyses achievable using the provided libraries, encompassing various algorithms across different domains and under diverse performance metrics. Recognizing the potentially high cost associated with training MORL algorithms, 10 runs of each algorithm were performed on each environment. It is important to note that while conducting experimental results with tighter confidence intervals is possible by increasing the number of runs, our main focus here is to highlight the capabilities of the presented libraries rather than conducting a comprehensive evaluation of all existing algorithms. Nevertheless, given that ORLB is publicly available and open to contributions, new users have the opportunity to contribute additional runs to enhance the statistical confidence of the results.

The hyperparameters used in each run can be accessed through the corresponding W&B dashboards. The runs detailed in this section were executed on high-performance computers at the University of Luxembourg [Varrette et al., 2014] and Vrije Universiteit Brussel. Training all algorithms across all environments, employing various random seeds, necessitated approximately 3 months of computation time. Examples of the W&B user interface, showcasing how our benchmark dataset can be analyzed, are provided in Figure 4.2 and Appendix B.[3] It is worth noting that some algorithms were trained over more timesteps than others, as they may exhibit differences in computational speed and sample efficiency. For instance, PGMORL relies on PPO [Schulman et al., 2017], which is computationally faster but less sample efficient than GPI-LS, which relies on TD3 [Fujimoto et al., 2018].

Figures 4.3, 4.4, and 4.5 illustrate the performance of various MORL-Baselines algorithms when assessed on MO-Gymnasium environments. Specifically, they display the mean and corresponding 95% confidence intervals concerning the evaluation metrics defined in Section 2.2.3. Additional results and plots can be found in our conference paper Felten et al. [2023a].

Figure 4.3 illustrates the performance of various MORL algorithms that support continuous action spaces in the *mo-halfcheetah-v4* domain. In this domain, GPI-LS exhibits strong performance in terms of expected utility. Regarding hypervolume, MORL/D SB PSA appears to perform best toward the end of the training phase. CAPQL and PGMORL display weaker performance for hypervolume and expected utility but seem to excel in terms of sparsity. Upon examining the PF of the best runs in terms of hypervolume, a similar observation as described in Section 3.2 can be made: CAPQL and PGMORL identify clustered points on only one end of the

[3] An overview of the command lines used to conduct each experiment is available at https://github.com/LucasAlegre/morl-baselines/issues/43.

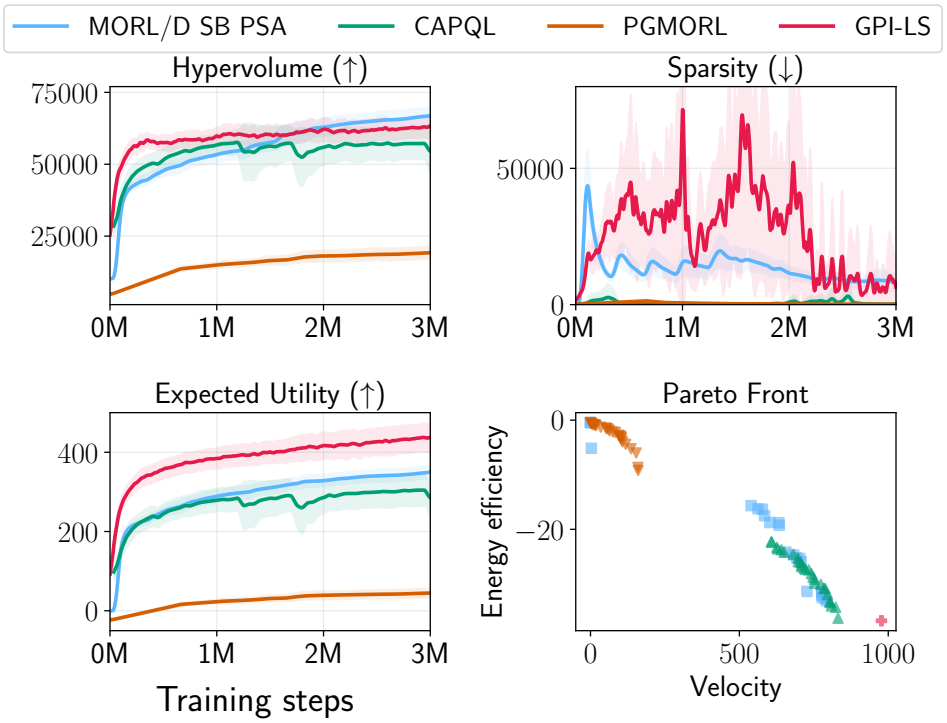
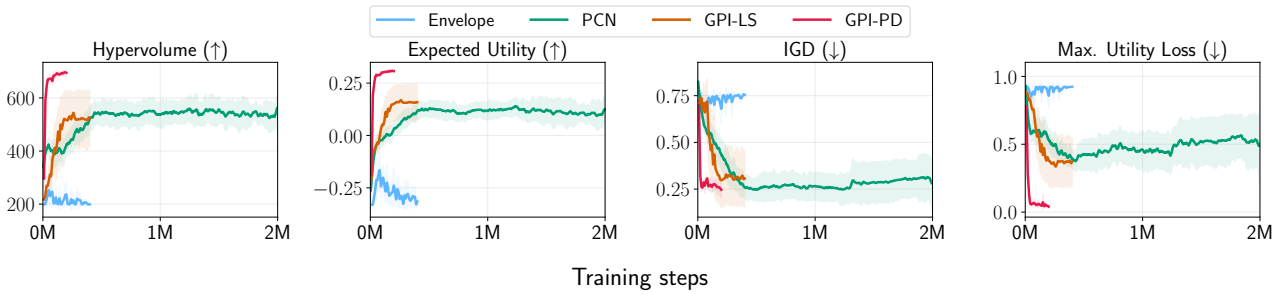


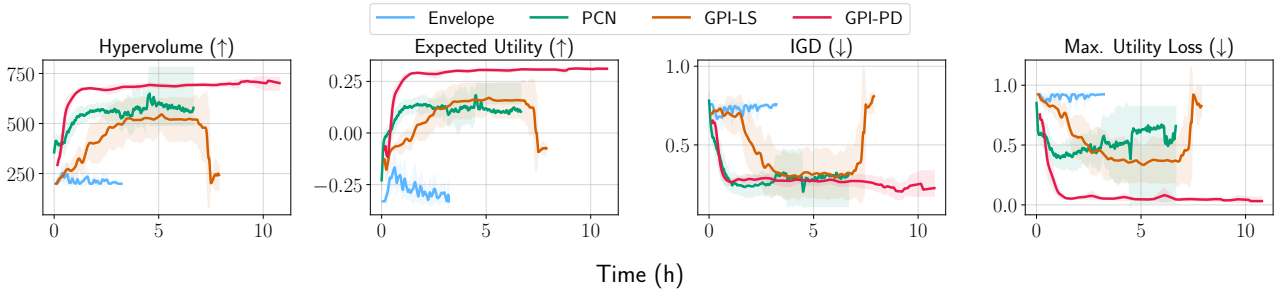
FIGURE 4.3: Peformance of MORL algorithms on the *mo-halfcheetah-v4* domain. Pareto fronts were constructed by identifying (across all runs) the front with the highest hypervolume after a given training budget (5M steps for PGMORL, 3M for the others).

objective space, resulting in a good sparsity metric while the found points fail to cover the entire objective space. Another notable observation is that GPI-LS only discovers a single policy maximizing the velocity objective. This solitary point yields a higher expected utility value compared to having multiple points with slightly worse policies in terms of convergence. We suspect GPI-LS learns only one point because it does not normalize the rewards and employs a conditioned network to encode all policies, thereby maximizing a scalarized value by learning a robust policy for the objective with the largest scale. Nevertheless, this underscores the importance of examining multiple metrics and the PF whenever feasible when comparing MORL algorithms.

While it remains a crucial factor for real-life deployments, the performance of algorithms in terms of wall-time is seldom reported in RL research papers, as they typically prioritize sample efficiency. However, this bias towards emphasizing on sample efficiency in papers can lead to computationally efficient methods being overlooked. To address this issue, some authors have begun reporting performance not only in terms of samples but also in terms of wall-time [Huang et al., 2024]. Our suite of libraries also facilitates reporting the performance of MORL algorithms in both cases as it records both the current sample number and wall-time for each new data. Figures 4.4(a) and 4.4(b) present different perspectives when considering different units on the x-axis on the *minecart* environment. Despite taking longer to execute, GPI-PD demonstrates strong results on this environment, surpassing all other algorithms in terms of hypervolume, expected utility, and maximum utility loss. Although designed for deterministic environments and applied to a stochastic environment, PCN shows promising results on this environment. Importantly, contrasting conclusions arise when comparing GPI-LS to PCN in terms of sample and wall-time efficiency in terms of hypervolume, IGD, and MUL: GPI-LS proves to be more sample efficient, whereas PCN's computational efficiency enables it to take significantly more steps per second, resulting in superior results for the same duration on fast environments. Lastly, Envelope appears to struggle to learn on this environment. The subsequent section of this thesis will delve into this case in more detail.



(A) **Sample efficiency:** Performance of MORL algorithms supporting discrete actions w.r.t. steps.



(B) **Wall-time efficiency:** Performance of MORL algorithms supporting discrete actions w.r.t. time.

FIGURE 4.4: Difference between sample efficiency and wall-time efficiency on *minecart-v0* for various algorithms. The total training budget was 200K steps for GPI-PD, 400K for GPI-LS and Envelope, and 2M for PCN.

Finally, Figure 4.5 illustrates the performance of various tabular MORL algorithms available on MORL-Baselines, evaluated across different deterministic environments. These types of plots are valuable for gaining an overview of various environments simultaneously. Specifically, we compare Pareto-based MORL techniques, such as PQL, with decomposition-based ones, such as MPMOQL. The MPMPOQL variants employ different weight generation techniques (random, OLS, or GPI-LS). Additionally, the GPI variant allows for combining learned policies during the evaluation phase. In these environments, PQL demonstrates very strong results, learning the optimal PF in all environments (indicated by MUL and IGD reaching 0). However, decomposition-based techniques, relying on linear scalarization, fail to discover the full Pareto front in the *deep-sea-treasure-concave* environment (see Section 2.2.5.1). On the *fruit-tree* environment, GPI-LS exhibits good performance compared to OLS and random weight generation methods.

4.3.2 Summary

This section presented the results of training MORL-Baselines algorithms on MORL-Gymnasium environments. The resulting data are hosted in ORLB [Huang et al., 2024], allowing for easy comparison and removing the need to rerun the baselines for new publications. Yet, it is worth noting that most of the combination of MORL-Baselines algorithm with MO-Gymnasium environment were not presented in the papers introducing the algorithms. Hence, these algorithms were not fine-tuned for all environments. For most runs, we kept the default hyperparameters which are based on the original papers. However, in certain instances where we could not achieve satisfactory results, we made a minor attempt to fine-tune the hyperparameters by hand. The next section dives deeper into the problem of hyperparameter optimization for MORL.

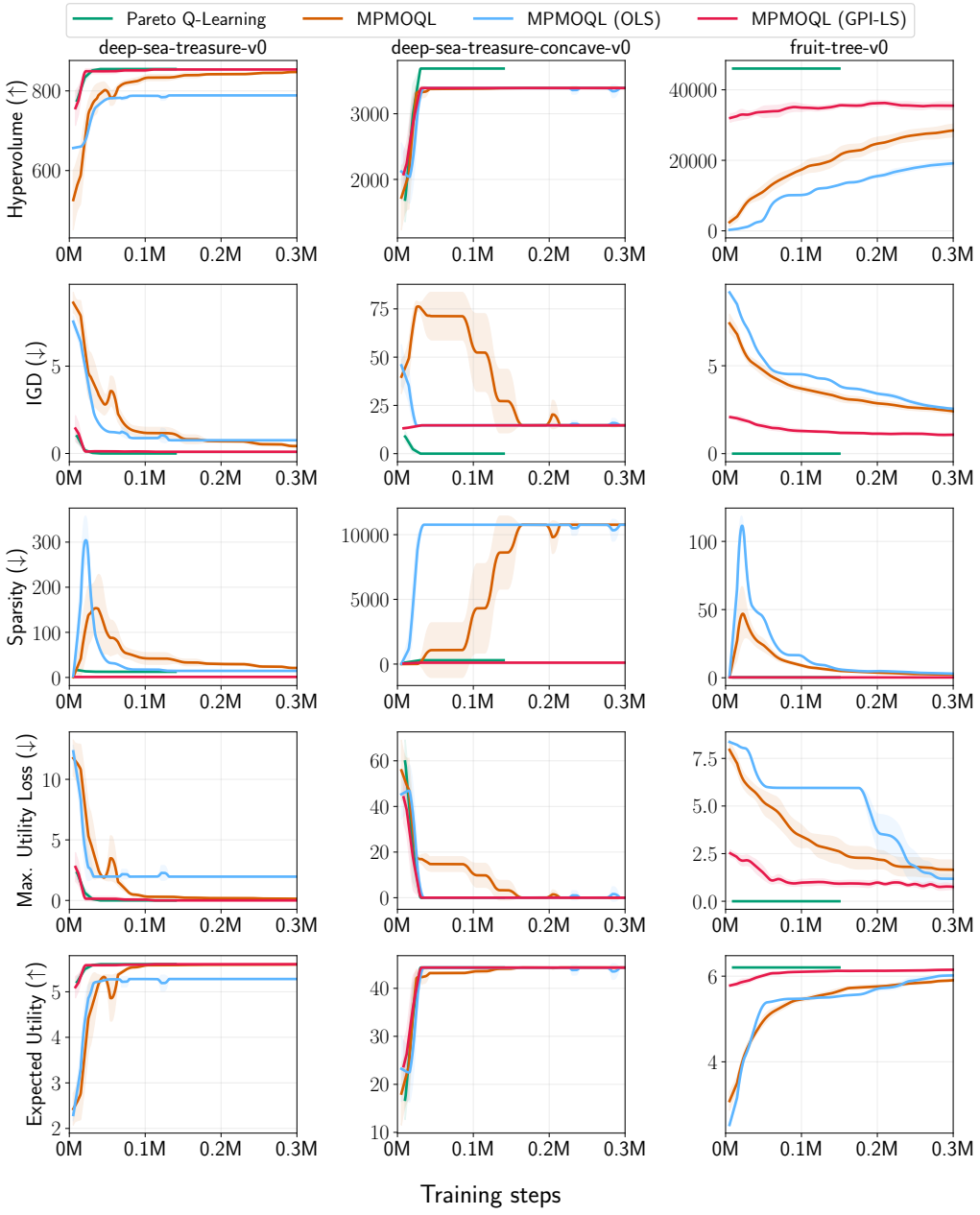


FIGURE 4.5: Performance of tabular MORL algorithms on different MORL environments w.r.t. training samples.

4.4 Hyperparameter Optimization for MORL

This section addresses the critical issue of optimizing hyperparameter values to enhance the performance of MORL algorithms and explores automated methods to achieve this goal. Despite the accessibility of open-source implementations of environments, algorithms, and tools, which have been highlighted in the preceding sections to encourage the adoption of MORL techniques even among non-domain experts, it is widely acknowledged that the effectiveness of RL (and MORL) agents heavily relies on the selection of appropriate hyperparameters [Henderson et al., 2018, Andrychowicz et al., 2021, Patterson et al., 2023]. This recognition underscores the reality that in novel environments, default hyperparameters may lead to suboptimal performance for existing algorithms. Additionally, comparisons between new algorithms and state-of-the-art techniques often suffer from the unfairness of pitting untuned baseline algorithms against finely tuned ones [Patterson et al., 2023]. This section aims to tackle issue 4, inadequate hyperparameter tuning, enumerated previously.

While manual hyperparameter tuning may initially seem attractive, the multitude of parameters and the time-intensive nature of obtaining training results render manual optimization impractical [Parker-Holder et al., 2022]. Consequently, having an automated approach to tuning hyperparameters for MORL, given a predefined level of effort, not only facilitates fairer comparisons within the field but also streamlines the implementation of these techniques for real-world applications. Despite recent efforts to automate such decisions in the realm of RL [Parker-Holder et al., 2022, Eimer et al., 2023], there has been a notable absence of similar endeavors concerning MORL. Moreover, existing solutions tailored for single-objective RL do not seamlessly translate to the multi-objective setting.

To address this gap, this part of the dissertation formally acknowledges the challenge of hyperparameter optimization for MORL and proposes a systematic approach to tackle it. Section 4.4.1 provides the necessary background knowledge and notation pertaining to hyperparameter optimization (HPO), while Section 4.4.2 formally defines the problem of hyperparameter optimization for multi-objective RL and introduces a methodology to address it. Lastly, Section 4.4.3 demonstrates the outcomes of applying the proposed methodology to a state-of-the-art method.

4.4.1 Hyperparameter Optimization for RL

Hyperparameter optimization for RL is the process of finding a set of hyperparameter values for the RL algorithm which leads to improved learning performances. The works of Parker-Holder et al. [2022], Eimer et al. [2023] provide formal definitions of the objectives in hyperparameter optimization for single-objective RL. In general, for a given RL algorithm denoted A, this problem can be defined as:

$$\max_{\boldsymbol{\psi}} \quad f(\boldsymbol{\psi}, \pi^*, v^{\pi^*}) \tag{4.1}$$

$$\text{s.t.} \quad \pi^*, v^{\pi^*} = \mathrm{A}(\boldsymbol{\psi}), \tag{4.2}$$

$$\boldsymbol{\psi} \in \Psi \wedge \mathrm{valid}(\boldsymbol{\psi}), \tag{4.3}$$

where $\boldsymbol{\psi}$ denotes the hyperparameter values, Ψ denotes the space of hyperparameter values, π^* and v^{π^*} denote the (potentially optimal) policy found by the RL algorithm and its corresponding value. f denotes the objective function for the result of the learning process using the given hyperparameters.

Aside from finding hyperparameter values leading to good performance, it is interesting to remark that the data collected during the optimization process may be useful in understanding the behavior of the studied RL algorithm. Indeed, the search memory can be seen as a dataset where features consist of hyperparameters and labels are the resulting objective values. From this, one can study the relative importance of hyperparameters on the final performances using sensitivity analysis tools from supervised learning [Bischl et al., 2023].

While the mathematical definition of the problem seems straightforward, solving such an optimization problem is in practice very challenging due to the following aspects.

Stochastic. The effectiveness of RL algorithms can vary based on the random seeds employed during training [Sutton and Barto, 2018, Henderson et al., 2018, Eimer et al., 2023]. As a result, aggregated metrics are often utilized to report these performances across multiple runs for statistical purposes. To achieve this in the

HPO context, the RL algorithm needs to be executed on various seeds for each possible set of hyperparameter values and aggregated values of the resulting performance can be used as an objective function. Formally,

$$f(\boldsymbol{\psi}) = \text{Agg}_{\sigma \in \Sigma_{\text{search}}} \Big(\text{Eval}(\text{A}, \boldsymbol{\psi}) \Big), \tag{4.4}$$

where $\text{Agg}_{\sigma \in \Sigma_{\text{search}}}$ represents an aggregation operator over a set of random seeds[4] used in the search phase (typically the mean), and Eval gives the objective value of training one policy given the hyperparameter values. In general, this evaluation is linked to the value of the returned policy, *e.g.* $\text{Eval}(\text{A}, \boldsymbol{\psi}) = v^{\pi^*}$, but more complex schemes exist and are discussed below.

Moreover, it is also crucial to ensure that the seeds utilized during the search phase are distinct from those employed for the final validation of the best hyperparameters discovered [Eimer et al., 2023], *i.e.*, $\Sigma_{\text{search}} \cap \Sigma_{\text{validation}} = \emptyset$. This precautionary measure prevents the optimization process from overfitting the search seeds.

Multi-objective. As mentioned above, the HPO objective function f can be different depending on the context; it is in general linked to the agent performance on one environment, but it could also be the total training time required, or the performance of a policy on different environments if the user wants to optimize for policies able to generalize, *e.g.*, $\text{Eval}(\text{A}, \boldsymbol{\psi}) = \mathbb{E}_{e \sim \mathcal{E}} \left[v^{\pi^*}(e) \right]$, with $\mathcal{E}$ representing a distribution of environments. Although most works choose to optimize only one objective for simplicity, some have already identified HPO for machine learning and RL to be multi-objective problems [Jin and Kacprzyk, 2006, Horn and Bischl, 2016, Binder et al., 2020, Parker-Holder et al., 2022, Bischl et al., 2023].

Constrained. There is a potential for erratic behaviors and exceptions to occur during the RL learning process due to various combinations of hyperparameter values. To represent this aspect of the problem, Equation 4.3 in the above optimization problem formulation establishes a space of configurations Ψ that is then checked for validity. In practice, these configurations can be verified either within the optimizer or within the RL algorithm itself, for example by returning a penalized objective value if an invalid configuration is detected.

Computationally expensive. It is widely acknowledged that training an RL algorithm on a specific environment demands a significant amount of computing time. Additionally, it is important to note that during the HPO process, the RL algorithm needs to be executed with multiple random seeds for each potential combination of hyperparameter values. This practical requirement renders the problem impractical for exhaustive search methods.

In practice, such expensive problems are usually solved by relying on Bayesian Optimization (BO) [Močkus, 1975, Jones et al., 1998], where a *surrogate* model of the objective value is learned by the optimizer based on the collected samples during the search process. This guides the optimizer to choose the most promising areas of the hyperparameter space. Moreover, early-stopping mechanisms such as hyperband [Falkner et al., 2018, Li et al., 2018] help allocate more resources to well-performing hyperparameters to better use the budget given to the optimization process.

[4] To avoid confusion with the RL states, the seeds are denoted using sigma.

4.4.2 Hyperparameter Optimization for MORL

Building on HPO for RL, this section presents the problem of hyperparameter optimization applied to the field of MORL. The main difference with HPO for RL lies in the fact that multi-policy MORL algorithms return a Pareto set and its corresponding Pareto front. Hence, Equations 4.1–4.3 can be adapted as follows:

$$\max_{\boldsymbol{\psi}} \quad f(\boldsymbol{\psi}, \mathcal{PS}, \mathcal{F}) \tag{4.5}$$

$$\text{s.t.} \quad \mathcal{PS}, \mathcal{F} = \mathrm{A}(\boldsymbol{\psi}) \tag{4.6}$$

$$\boldsymbol{\psi} \in \Psi \wedge \mathrm{valid}(\boldsymbol{\psi}), \tag{4.7}$$

Dealing with sets as output entails an additional step in the HPO process: transforming the PFs into scalars for comparison. As discussed in Section 2.2, there are two aspects (or objectives) to consider in such a case: *convergence* and *diversity*. This means that the problem of HPO for MORL is also multi-objective. Similar to Pareto-based optimization methods (Section 2.2.4), a simple solution to such an issue is to rely on performance metrics. In particular, hybrid performance metrics can quantify both aspects at the same time. Yet these metrics also suffer from their ability to summarize the information; for example, it is possible to have a large hypervolume score with only one well-placed point in the PF, while a PF composed of many points may have a small hypervolume score (see GPI-LS in Figure 4.3).

4.4.2.1 Proposed Methodology

Building on all these aspects and extending the work of Eimer et al. [2023], we argue that the problem of HPO for MORL can be solved in two phases: a first phase which searches for hyperparameter values on a range of searching seeds, and a second phase which validates the final performance of the found hyperparameters on a range of validation seeds.

Algorithm 4.1 illustrates our proposed methodology. The algorithm starts by instantiating an optimizer, which is in charge of proposing hyperparameter values and keeping track of the associated final performances (line 1). Such an optimizer can be based on BO such as Falkner et al. [2018], but also could rely on simpler mechanisms such as grid search or random search. Next, the algorithm enters its search phase for a given amount of time specified by a stopping criterion. For each trial, the optimizer generates a new hyperparameter values candidate (line 3). These hyperparameters are then used to train the MORL algorithm on various search seeds for a given budget b_{search} (lines 4–6). As outlined above, a key distinction from HPO for RL emerges. In the MORL scenario, the outcomes of diverse runs manifest as PFs (sets of vectors), necessitating conversion into comparable entities. Consequently, each of these PFs is initially converted into scalar values through the utilization of a performance metric, akin to how fitness measures are used in Pareto-based optimization techniques (Section 2.2.4). Next, to deal with the stochastic aspect of the optimization problem, these result metrics are aggregated as discussed in Section 4.4.1 and reported to the optimizer (lines 7–8). Then, the hyperparameters leading to the best evaluation are returned by the optimizer to start the validation phase (line 10). In this phase, the MORL algorithm is trained over various validation seeds for a given budget $b_{\text{validation}}$ (lines 11–13). Finally, the achieved PFs and best-performing hyperparameters are returned to the user for final inspection.

Algorithm 4.1 Hyperparameter optimization for MORL.

Input: Stopping criterion *stop*, MORL algorithm A, hyperparameter space Ψ, environment *env*, searching budget b_{search}, validation budget $b_{\text{validation}}$, optimizer *Opt*, set of search seeds Σ_{search}, set of validation seeds $\Sigma_{\text{validation}}$.

Output: The best hyperparameter values for algorithm A on *env*, $\boldsymbol{\psi}^*$ and corresponding Pareto fronts.

$opt = new\ Opt(\Psi)$ ▷ Searching phase
while $\neg stop$ **do**
  $\boldsymbol{\psi} = opt.next()$
  for $\sigma \in \Sigma_{\text{search}}$ **do**
    $\mathcal{F}_\sigma, \mathcal{PS}_\sigma = A(\boldsymbol{\psi}, \sigma, b_{\text{search}}, env)$
  end for
  $f = Agg_{\sigma \in \Sigma_{\text{search}}}(Eval(\mathcal{F}_\sigma))$ ▷ See Equation 4.4
  $opt.report(\boldsymbol{\psi}, f)$
end while
$\boldsymbol{\psi}^* = opt.best()$ ▷ Validation phase
for $\sigma \in \Sigma_{\text{validation}}$ **do**
  $\mathcal{F}_\sigma, \mathcal{PS}_\sigma = A(\boldsymbol{\psi}^*, \sigma, b_{\text{validation}}, env)$
end for
return $\mathcal{F}, \boldsymbol{\psi}^*$

Hyperparameter	**Value Range**
`learning_rate`	$[0.001, 0.0001]$
`learning_starts`	$[1..1,000]$
`buffer_size`	$[1,000..2,000,000]$
`max_grad_norm`	$[0.1, 10.0]$
`gradient_updates`	$[1..10]$
`target_net_update_freq`	$[1..10,000]$
`tau`	$[0.0, 1.0]$
`num_sample_w`	$[2..10]$
`per_alpha`	$[0.1, 0.9]$
`initial_homotopy_lambda`	$[0.0, 1.0]$
`final_homotopy_lambda`	$[0.0, 1.0]$
`homotopy_decay_steps`	$[1..100,000]$
`initial_epsilon`	$[0.01, 1.0]$
`final_epsilon`	$[0.01, 1.0]$
`epsilon_decay_steps`	$[1..100,000]$

TABLE 4.2: Hyperparameters search space (Ψ).

4.4.3 Experiments

This section presents the results obtained when applying the presented methodology to parameterize a state-of-the-art MORL algorithm. Our main goals are: (1) to validate that HPO provides an advantage compared to relying on default hyperparameter values, and (2) to showcase the potential benefits of studying hyperparameters through the lens of sensitivity analysis on the search memory.

To conduct our experiments, we optimized 15 hyperparameters of Envelope Q-Learning [Yang et al., 2019] on a benchmark problem from MO-Gymnasium. The hyperparameter space described in Table 4.2 was used to perform

Parameter	Value
Ψ	See Table 4.2
stop	48 hours (37 trials)
Σ_{search}	[10..12] (3 seeds)
b_{search}	100,000 steps
Eval	HV($\mathcal{F}_\sigma$), $\mathbf{z}_{\text{ref}}$ = (-1,-1,-200)
Agg	Mean
$\Sigma_{\text{validation}}$	[0..9] (10 seeds)
$b_{\text{validation}}$	400,000 steps

TABLE 4.3: HPO parameters for our experiments.

the optimization. Having such a large number of hyperparameters to consider makes it virtually impossible for a human to tackle the problem. In this context, we study Envelope on the *minecart-v0* environment [Abels et al., 2019] (Figure 4.1, left), which was shown to be challenging for this algorithm in Figure 4.4. In this environment, the agent controls a cart and has three objectives: it must collect two types of ore from different mines as well as minimize fuel consumption. Finding a diverse PF in such a domain is particularly challenging due to the delayed rewards associated with both ores. Indeed, the agent must first go to a mine to gather ore, then return to its base to obtain the related rewards. This necessitates a significant amount of exploration.

Our experiments were conducted using the implementations in MORL-Baselines, MO-Gymnasium, and we used the Bayesian optimizer of W&B [Biewald, 2020]. Table 4.3 presents a global overview of the parameters linked to the optimization process.[5] Our code has been included into MORL-Baselines and can be found there. To tackle the stochastic aspect of the problem, we chose to run each trial on 3 search seeds (in parallel) in order to balance between speed and accuracy of the trials' estimates. Despite its limitations, hypervolume has been chosen as performance metric because of its ability to characterize both diversity and convergence. The total optimization budget was 2 days, which was equivalent to running 37×3 training processes (each one for 100,000 steps). For reference, each validation training (400,000 steps) required approximately 3.5 hours. All our experiments have been conducted using the high-performance computer of the University of Luxembourg [Varrette et al., 2014].

4.4.3.1 Sensitivity Analysis

The integration with W&B provides dashboards allowing the analysis of the search phase (see Figure 4.6). These dashboards illustrate the objective values found during the search process over time (top left). They also show a parallel plot displaying the hyperparameter values and their resulting objective value (bottom). Finally, in the top right corner, they utilize the search memory as a dataset, where hyperparameter values serve as features and resulting performances as labels. Then, a random forest is trained to calculate parameter importance, while linear correlations between hyperparameters and final results are also presented. In the case under study, we extracted the four most significant hyperparameters from the dashboard and listed these in Table 4.4.

The table indicates that the homotopy optimization approach used by Envelope has a strong influence on the final results. This involves optimizing a linear combination of two loss functions to smoothen the objective landscape, given by $L(\theta) = (1 - \lambda_k) \cdot L^{\mathtt{A}}(\theta) + \lambda_k \cdot L^{\mathtt{B}}(\theta)$, where $L^{\mathtt{A}}$ is a loss based on vectorial Q-values and $L^{\mathtt{B}}$ is based on scalarized values. The parameter $\lambda_k \in$ [`initial_homotopy_lambda`, `final_homotopy_lambda`] is increased throughout the learning process based on a schedule given by the parameters listed in the table. It seems that on this environment, a large λ (and thus relying more on $L^{\mathtt{B}}$) is correlated with good performances.

[5] The W&B hyperparameter optimization dashboard can be found at: https://api.wandb.ai/links/openrlbenchmark/9ffv3e5c

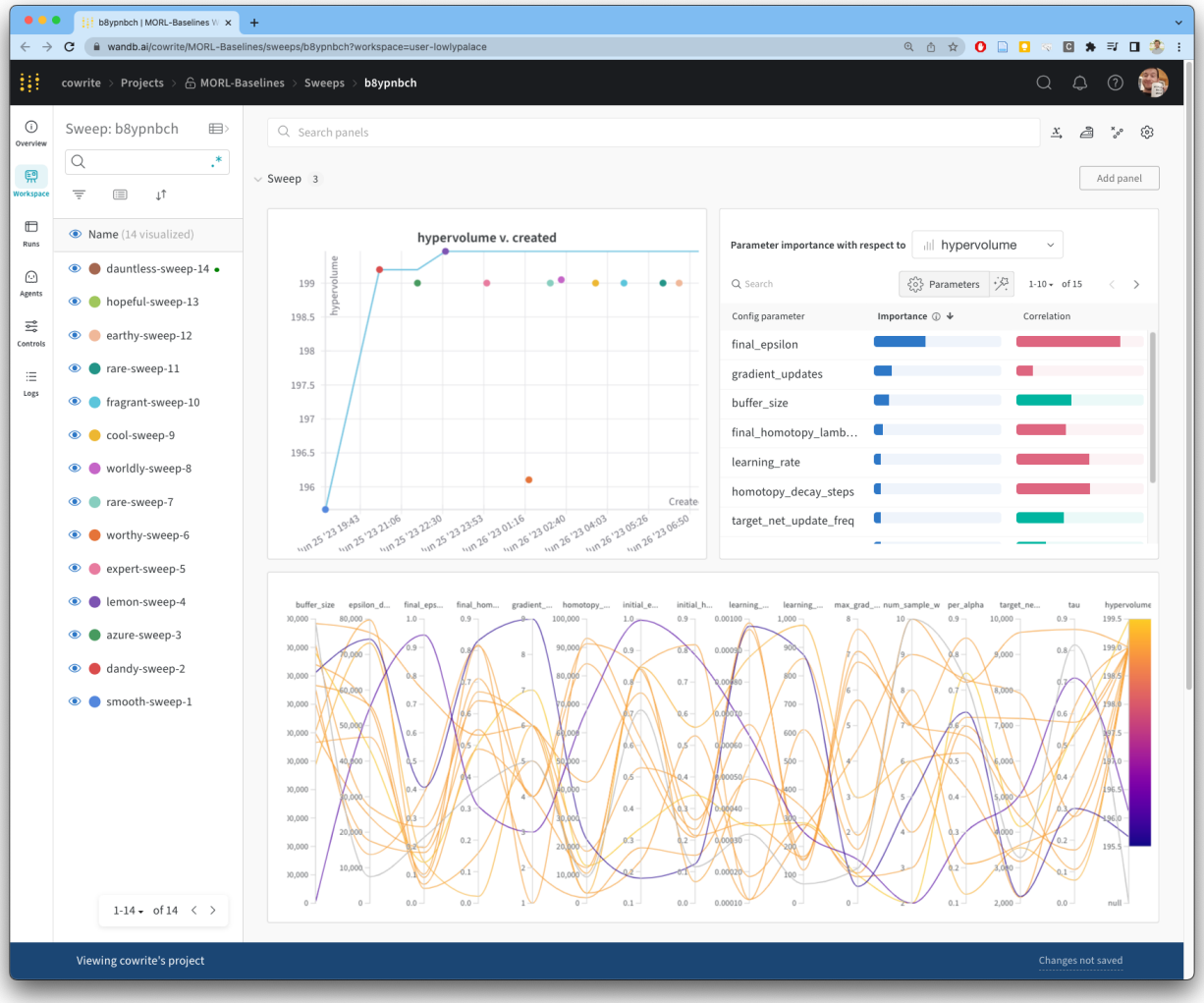
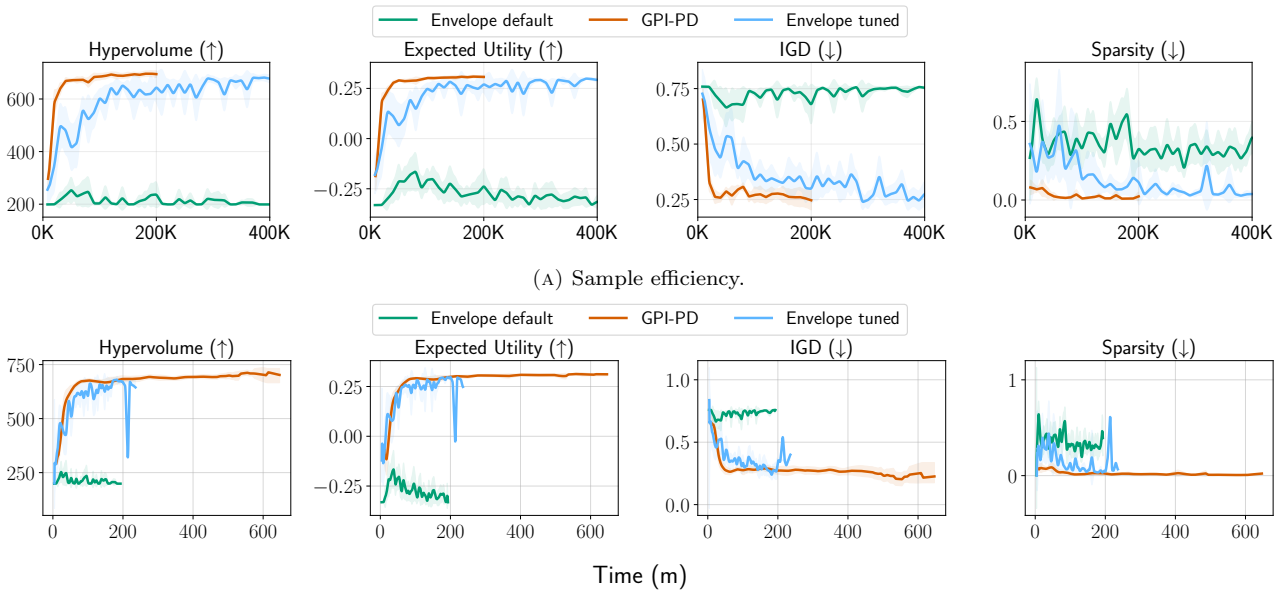


FIGURE 4.6: Dashboard of hyperparameter search results, with panel for analysis of parameter importance and correlation.

Parameter	Importance	Correlation
`final_homotopy_lambda`	0.288	0.678
`tau`	0.222	-0.617
`homotopy_steps`	0.101	0.558
`epsilon_decay_steps`	0.064	0.457

TABLE 4.4: Hyperparameters importance and correlations.



(A) Sample efficiency.

(B) Wall-time efficiency.

FIGURE 4.7: Average performance metrics and 95% confidence intervals over 10 seeded runs of the tuned algorithm and existing baselines on *minecart-v0*.

The parameter `tau` is linked to the soft update of the target network, as in DQN [Mnih et al., 2015]. Keeping this parameter relatively low appears to be linked to better performance. Finally, `epsilon_decay_steps` regulates the decay rate of the agent's exploration strategy. Maintaining a high level of exploration appears to be associated with favorable final performance. This observation is consistent with the earlier discussion concerning the environment's delayed rewards. For a more detailed explanation of these parameters, refer to the original Envelope paper by Yang et al. [2019].

4.4.3.2 Validation

To validate the effectiveness of the proposed method, we compare the results of training Envelope with the tuned hyperparameters (Envelope tuned) against training with the default hyperparameters (Envelope default) defined in MORL-Baselines. Furthermore, to establish an upper baseline, we compare the results against GPI-PD [Alegre et al., 2023], one of the most sample-efficient MORL algorithms available at the time of writing. For the baseline results, we reuse the ones provided by MORL-Baselines in ORLB [Felten et al., 2023a, Huang et al., 2024]. To compare these algorithms, we rely on the performance metrics discussed in Section 2.2.3. These allow for assessing the diversity and convergence aspects of the resulting PFs as well as the final expected user utility.

Figures 4.7 illustrate the learning curves of the three different algorithms against training steps and wall-time. There is a drastic difference between Envelope default and its tuned counterpart. Indeed, as previously discussed, Envelope default seems to fail to discover new policies throughout the learning process, as all metrics stay rather flat. On the other hand, Envelope tuned improves on all metrics, meaning it improves the learned PF. Upon comparison with GPI-PD, Envelope tuned exhibits remarkably strong performance, achieving results almost on par with this more recent technique in terms of sample efficiency, and on par in terms of wall-time efficiency. Naturally, to ensure a fair comparison between Envelope and GPI-PD, the latter algorithm should also be tuned on this environment. Nevertheless, even untuned, GPI-PD gives a strong baseline for comparison. These results show the crucial importance of carefully tuning hyperparameters when deploying MORL algorithms on new environments.

4.4.4 Summary

In this section, we discussed the benefits and challenges of hyperparameter optimization for multi-objective reinforcement learning. We presented the existing literature on MORL and HPO for RL. Then, we derived a methodology for conducting HPO for MORL based on these. As early results, our methodology has been applied on a state-of-the-art algorithm and shown to greatly improve its final performances on a classical MORL environment. We also presented an analysis of the importance of some hyperparameters using the newly introduced tools.

4.5 Conclusion

This part of the thesis presented contributions aimed at facilitating the practical evaluation and deployment of MORL agents. Firstly, MO-Gymnasium introduced a standard API and a set of environments. This library streamlines researchers' tasks by providing a diverse collection of domains for algorithm designers to test their contributions against and compare them with the state of the art. Secondly, MORL-Baselines presented a collection of reliable MORL algorithm implementations. This library is compatible with the MO-Gymnasium API, enabling non-domain experts to apply the implemented solving methods to new environments. Additionally, the project offers dependable implementations and tools to simplify the work of MORL researchers. Thirdly, all MORL-Baselines implementations have been executed on MO-Gymnasium environments, and the outcomes of these training processes have been uploaded to a public dataset in Open RL Benchmark [Huang et al., 2024]. This facilitates proper tracking of experimental details such as code version and hyperparameter values. Moreover, leveraging this public dataset, MORL researchers do not need to rerun the baselines when comparing their algorithms with the state of the art. This significantly reduces the cost of research, both in terms of time and energy. Finally, the last contribution in this part is a study of hyperparameter optimization for MORL. Particularly, we proposed an automated method to discover hyperparameter values that lead to good performance of MORL algorithms. This enables easier deployment of MORL in real-world, unseen scenarios, as well as fairer comparisons in the research community.

MORL shows significant promise for real-world deployment, particularly in contrast to RL's reliance on *a priori* scalarized methods and the often tedious trial-and-error weight-finding process. However, there are situations where multiple agents interact within the same environments, or where the number of actions is excessively large. Such scenarios are typically addressed with multi-agent RL with *a priori* scalarization and trial-and-error to determine the desired weight vector. We believe this could be improved by progress in multi-objective multi-agent RL. However, MOMARL has not been extensively studied yet. The next part of the thesis studies the extension of MORL techniques to environments containing multiple agents, taking initial steps to address this gap in research.

Published Work

MO-Gymnasium was originally published as MO-Gym at the 34th Benelux Conference on Artificial Intelligence (BNAIC):

- Lucas N. Alegre, Florian Felten, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, Ana LC Bazzan, and Bruno C. da Silva. MO-Gym: A Library of Multi-Objective Reinforcement Learning Environments. In *Proceedings of the 34th Benelux Conference on Artificial Intelligence BNAIC/Benelearn*, 2022

The contributions described in Sections 4.1–4.3 were published in the proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS):

- Florian Felten, Lucas Nunes Alegre, Ann Nowe, Ana L. C. Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno Castro da Silva. A Toolkit for Reliable Benchmarking and Research in Multi-Objective Reinforcement Learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023a

The contributions described in Section 4.4 were peer-reviewed and presented at the 2nd Multi-Objective Decision Making Workshop (MODeM):

- Florian Felten, Daniel Gareev, El-Ghazali Talbi, and Grégoire Danoy. Hyperparameter Optimization for Multi-Objective Reinforcement Learning, October 2023b. URL http://arxiv.org/abs/2310.16487. arXiv:2310.16487 [cs]

Part III

Multi-Objective Multi-Agent Reinforcement Learning

Unity makes strength.

— *Belgian national motto*

Chapter 5

Building Reliable Foundations for MOMARL Research

Contents

The previous chapters of this thesis introduced various advances in the realm of single-agent multi-objective reinforcement learning. While these developments hold promise for deploying agents capable of making informed compromises, they fall short in addressing problems involving multiple agents as discussed in Section 2.4. Multi-objective multi-agent settings have received considerably less attention from the research community when compared to RL, MORL, or MARL. This is partly due to the fact that this field of study was only recently formally defined in the work of Rădulescu et al. [2020]. Previous to this seminal study, contributions were fractured and hard to identify. Moreover, problems were often simplified, ignoring either the multi-agent or multi-objective dimension of the studied problems. Consequently, there is currently little work on benchmarks and solving methods that recognize and take both aspects into account. To establish a solid foundation for this emerging field, we propose the introduction of tools, baselines, and challenges similar to those established in MORL in the last chapter.

In this chapter, we present MOMAland, the first collection of benchmark environments and utilities for MOMARL.[1] Essentially, the MOMAland APIs and environments can be seen as either extensions of PettingZoo [Terry et al., 2021] to the multi-objective setting, or extensions of MO-Gymnasium (Section 4.1) to the multi-agent setting. Additionally, we describe how to use existing solving methods from MORL and MARL to solve cooperative MOMARL problems when the decision-maker's preferences are unknown. Implementations of such methods are also included in MOMAland.

[1] Documentation: https://momaland.farama.org/.
Code: https://github.com/Farama-Foundation/momaland.

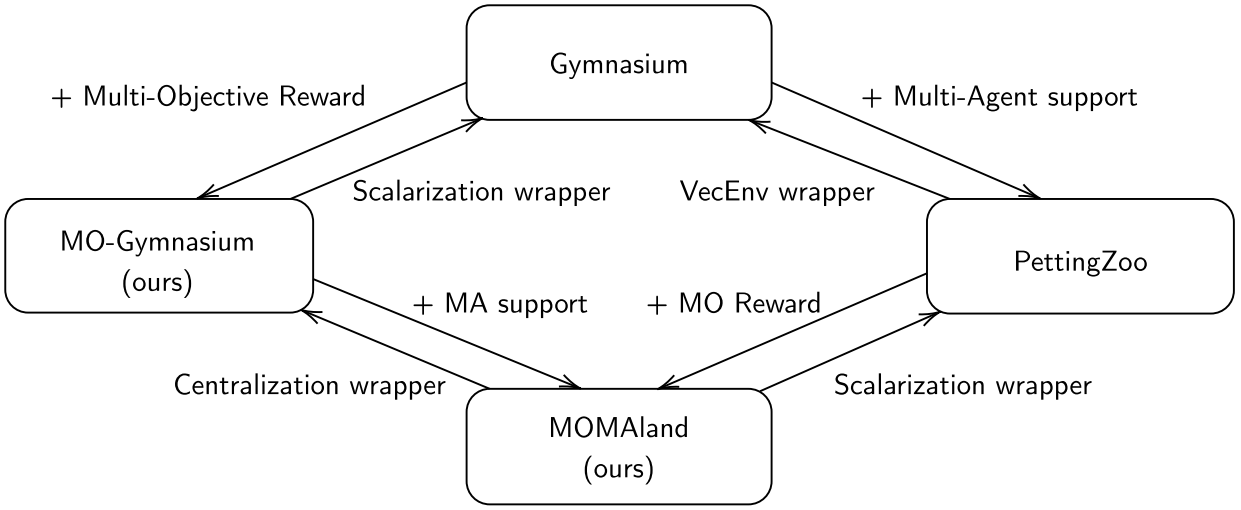


FIGURE 5.1: Overview of the libraries related to MOMAland within the Farama Foundation.

5.1 MOMAland: A Collection of Baselines for MOMARL

Many real-world problems such as smart electrical grids [Lu et al., 2022b], traffic systems [Houli et al., 2010], or infrastructure management planning [Leroy et al., 2023] involve controlling multiple components while making compromises among several conflicting objectives. Up to now, the multi-agent aspect has been well studied, and there is a wealth of literature on multi-agent approaches spanning several decades. Separately, over the last 20 years, there has been an increasing interest in single-agent multi-objective approaches, as presented in the previous chapters. However, the intersection of these two aspects, *multi-objective multi-agent reinforcement learning* (MOMARL) has not received much attention. Indeed, problems are often simplified by hard-coding a trade-off among objectives or centralizing all decisions in a single agent. We argue that many of the most complex problems have both a multi-objective and a multi-agent dimension, and therefore it is crucial to progress the field of MOMADM to enable future progress in many potential AI application areas.

As discussed in Section 4.1, the development of standardized benchmarks has been a key factor that drove progress in various areas of AI over the years. Without standardized, publicly available benchmarks, researchers spend a lot of unnecessary time re-implementing test environments from published papers, reproducibility is made much more difficult, and results published in different papers are potentially incomparable [Patterson et al., 2023, Felten et al., 2023a]. Suites of standardized benchmarks have helped to address these issues already in some fields of AI, such as the seminal Gymnasium library [Towers et al., 2023] for single-objective single-agent RL, the PettingZoo library [Terry et al., 2021] for multi-agent RL, and more recently, MO-Gymnasium [Alegre et al., 2022, Felten et al., 2023a] (Section 4.1) for multi-objective RL.

In this chapter, we introduce MOMAland, the first publicly available set of MOMADM benchmarks. MOMAland, incorporated within the Farama Foundation ecosystem (see Figure 5.1), draws inspiration from and is compatible with analogous projects and currently offers over 10 environments encompassing diverse MOMARL research settings. By embracing open-source principles and inviting contributions, we anticipate that MOMAland will evolve in tandem with research trends and host new environments in the future.

Additionally, MOMAland includes utilities and novel learning algorithms intended to establish foundational baselines for future research in MOMARL. Notably, it offers utilities enabling the utilization of existing MORL and MARL-solving methods through centralization or decomposition strategies. Importantly, while the provided

Domain	# of agents	# of objectives	stochastic transitions?	full observability possible?	partial observability possible?	team rewards possible?	individual rewards possible?	discrete/continuous state (d/c)	discrete/continuous actions (d/c)
MO-BPD	2-n	2	✗*	✗	✓	✓	✓	d	d
MO-ItemGathering	2-n	2-m	✗*	✓	✗	✗	✓	d	d
MO-GemMining	2-n	2-m	✗*	-	-	✗	✗	-	d
MO-RouteChoice	2-n	2	✗	-	-	✗	✓	-	d
MO-PistonBall	2-n	3	✗*	✗	✓	✗	✓	c	d/c
MO-MW-Stability	2-n	2	✗	✓	✓	✓	✓	c	c
CrazyRL/Surround	2-n	2	✗	✓	✗	✓	✓	c	c
CrazyRL/Escort	2-n	2	✗	✓	✗	✓	✓	c	c
CrazyRL/Catch	2-n	2	✓	✓	✗	✓	✓	c	c
MO-Breakthrough	2	1-4	✗	✓	✗	✗	✓	d	d
MO-Connect4	2	2-20	✗	✓	✗	✗	✓	d	d
MO-Ingenious	2-6	2-6	✗	✓	✓	✓	✓	d	d
MO-SameGame	1-5	2-10	✗*	✓	✗	✓	✓	d	d

TABLE 5.1: List of environments implemented in MOMAland. State observability and discreteness are not specified for MO-GemMining and MO-RouteChoice as these are stateless domains. Upper limits specified as "n" or "m" signal that the environment in question does not enforce an upper limit on the number of agents or objectives, respectively. (*) These environments can have randomized starting states, but otherwise no stochastic transitions.

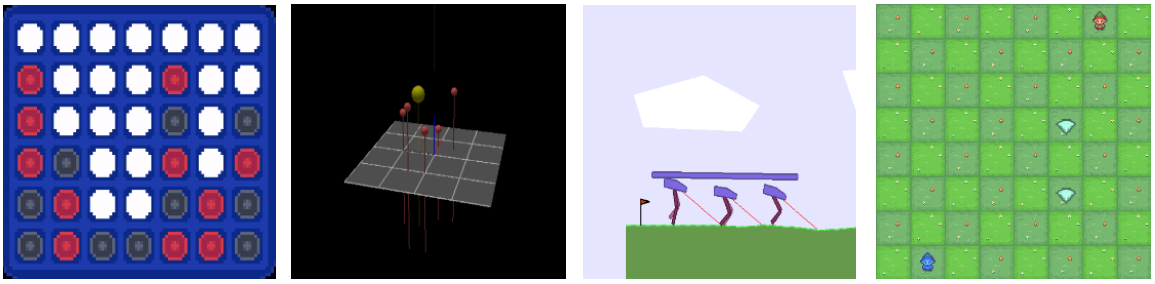


FIGURE 5.2: Visualization of some environments in MOMAland. From left to right: MO-Connect4, CrazyRL/-Surround, MO-MultiWalker-Stability, MO-ItemGathering.

baselines can find solutions for certain MOMADM settings, MOMAland also features environments with no known solution concept, *i.e.*, adversarial or mixed settings with unknown preferences. Addressing these environments requires tackling open research questions before deriving appropriate solving methods.

The rest of this chapter is organized as follows: Section 5.2 presents the environments currently included in our library, Section 5.3 illustrates the APIs exposed in MOMAland, Section 5.4 enumerates additional utilities and tools provided in the library, Section 5.5 presents algorithmic baselines designed to tackle some of the introduced environments and results are presented in Section 5.6. Finally, Section 5.7 summarizes our work.

5.2 Available Environments

MOMAland currently provides over 10 environments that offer a diverse range of possibilities to benchmark MOMARL algorithms. Figure 5.2 illustrates the rendering of some of the provided environments. Table 5.1 provides an overview of all environments, according to the criteria listed in the foundational survey of Rădulescu et al. [2020], which are essential for multi-objective multi-agent settings. These environments cover a spectrum of features, including discrete and continuous state and action spaces, stateless and stateful environments, cooperative and competitive settings, as well as fully and partially observable states. Notably, these last two criteria go beyond the scope of this thesis and are thus not developed in this manuscript, they are however developed in the journal paper introducing MOMAland [Felten et al., 2024b].

Some of these environments are multi-objective extensions of PettingZoo (*e.g.*, MO-MultiWalker is a multi-objective extension of Gupta et al. [2017]), others such as MO-ItemGathering [Källström and Heintz, 2019] have been taken from the current literature in MOMARL, and some are introduced in this work, *e.g.*, the CrazyRL environments. The environments are briefly described below. More information is available in the MOMAland paper [Felten et al., 2024b] and a complete list of supported environments is available on our documentation website: `https://momaland.farama.org/`.

Multi-Objective Beach Problem Domain (MO-BPD). The Multi-Objective Beach Problem Domain (MO-BPD) [Mannion et al., 2018] is a setting with two objectives, reflecting the enjoyment of tourists (agents) on their respective beach sections in terms of crowdedness and diversity of attendees. Each beach section is characterized by a capacity and each agent is characterized by a type. These properties, together with the location selected by the agents on the beach sections, determine the vectorial reward received by agents. The MO-BPD domain has two reward modes: (i) *local reward*, where each agent receives the reward signal associated with its respective beach section (selfish); and (ii) *global reward*, where the reward signal for each agent is an objective-wise sum over all the beach sections (cooperative).

MO-ItemGathering. The Multi-Objective Item Gathering domain (Figure 5.2, rightmost picture), adapted from Källström and Heintz [2019], is a multi-agent grid world, containing items of different colors. Each color represents a different objective and the goal of the agents is to collect as many objects as possible. MO-ItemGathering is fully observable and assigns individual rewards to the agents.

MO-GemMining. In Multi-Objective Gem Mining, extending Gem Mining / Mining Day [Bargiacchi et al., 2018, Roijers et al., 2015a, Roijers, 2016] to multiple objectives, a number of villages (agents) send workers to extract gems from different mines. Each gem type represents a different objective. There are restrictions on which mines can be reached from each village. Furthermore, workers influence each other in their productivity. MO-GemMining is stateless; each action corresponds to one independent mining day. It is fully cooperative and can be modeled as a multi-objective multi-agent multi-armed bandit.

MO-RouteChoice. MO-RouteChoice is a multi-objective extension of the route choice problem [Thomasini et al., 2023], where a number of self-interested drivers (agents) must navigate a road network. Each driver chooses a route from a source to a destination while minimizing two objectives: travel time and monetary cost. Both objectives are affected by the selected routes of the other agents, as the more agents travel on the same path, the higher the associated travel time and monetary cost. The number of agents is configurable. The environment contains various road networks from the original route choice problem [Ramos et al., 2020,

Thomasini et al., 2023], including Braess paradoxes and networks inspired by real-world cities. MO-RouteChoice is a stateless game, where each agent chooses one of the possible routes from its source to its destination and receives an individual reward based on the joint strategy of all agents.

MO-PistonBall. MO-PistonBall is based on an environment published in PettingZoo [Terry et al., 2021] where the goal is to move a ball to the edge of the window by operating several pistons (agents). In the original environment, the reward function is individual per piston and computed as a linear combination of three components. Concretely, the total reward consists of a global reward proportional to the distance to the wall, a local reward for any piston that is under the ball, and a per-timestep penalty. In the MOMAland adaptation, the environment dynamics are kept unchanged, but now each reward component is returned as an individual objective.

MO-MW-Stability. Multi-Objective Multi Walker Stability (Figure 5.2, third picture from the left) is another adaptation of a PettingZoo environment, originally published in Gupta et al. [2017], to multi-objective settings. In this environment, multiple walker agents aim to carry a package to the right side of the screen without falling. The multi-objective version of this environment includes an additional objective to keep the package as steady as possible while moving it. Naturally, achieving higher speed entails greater shaking of the package, resulting in conflicting objectives. This environment is cooperative and agents only have a partial view of the global state. Hence, it is a MODec-POMDP.

CrazyRL. CrazyRL (Figure 5.2, second picture from the left) consists of 3 novel 3D environments in which drones (agents) aim to surround a potentially moving target. The two objectives of the drones are to minimize their distance to the target while maximizing the distance between each other. The 3 environments differ in the behavior of the target, which can be static, move linearly, or actively try to escape the agents. These environments are cooperative and agents can perceive the location of everyone else. Hence, they are all MOMMDPs. Importantly, these environments are used in the next chapter to conduct real-world experiments with MOMARL-solving methods.

MO-Breakthrough. MO-Breakthrough is a multi-objective variant of the two-player, single-objective turn-based board game Breakthrough. In MO-Breakthrough there are still two agents, but up to three objectives in addition to winning: a second objective that incentivizes faster wins, a third one for capturing opponent pieces, and a fourth one for avoiding the capture of the agent's own pieces. The game is competitive and fully observable.

MO-Connect4. MO-Connect4 is a multi-objective variant of the two-player, single-objective turn-based board game Connect 4 (Figure 5.2, leftmost picture). In addition to winning, MO-Connect4 extends this game with a second objective that incentivizes faster wins, and optionally one additional objective per column of the board. As the board size is configurable, so is the number of these objectives. MO-Connect4 is competitive and fully observable.

MO-Ingenious. MO-Ingenious is a multi-objective adaptation of the zero-sum, turn-based board game Ingenious. The game's original rules support 2-4 players collecting scores in multiple colors (objectives), with the goal of winning by maximizing the minimum score over all colors. In MO-Ingenious, we leave the utility wrapper up to the users and only return the vector of scores in each color objective. MO-Ingenious has two reward

modes: (i) *individual reward*, where each agent receives scores only for their own actions; and (ii) *team reward*, where all collected scores are shared by all agents. Furthermore, it can be played with (i) *partial observability* as the original game, or in a (ii) *fully observable* mode.

MO-SameGame. MO-SameGame is a multi-objective, multi-agent variant of the single-player, single-objective turn-based puzzle game called SameGame. The original game rewards the player with points for every action, which the player tries to maximize. MO-SameGame extends this to a configurable number of agents, acting alternatingly, and a configurable number of different types of points (objectives) to be collected. MO-SameGame has two reward modes: (i) *individual reward*, where each agent receives points only for their own actions; and (ii) *team reward*, where all collected points are shared by all agents.

5.3 MOMAland APIs

```
1 from momaland.envs.multiwalker_stability import momultiwalker_stability_v0 as _env
2
3 env = _env.parallel_env(render_mode="human")
4 observations, infos = env.reset(seed=42)
5 while env.agents:
6     actions = {agent: policies[agent](observations[agent]) for agent in env.agents}
7     # vec_reward is a dict[str, numpy array]
8     observations, vec_rewards, terminations, truncations, infos = env.step(actions)
9 parallel_env.close()
```

Listing 2: Parallel API usage for simultaneous actions.

MOMAland exposes 2 APIs that extend PettingZoo's functionalities only where necessary, specifically in returning a vectorial reward (*i.e.*, a NumPy array [Harris et al., 2020]) instead of a scalar for each agent. Moreover, the specifications of the functions exposed in the APIs are very similar to the ones of MO-Gymnasium (Section 4.1), making it easier for users to understand the usage of both libraries.

The first API, referred to as *parallel*, enables all agents to act simultaneously, as demonstrated in Listing 2. In this mode, all actions are collectively provided to the step function as a dictionary, mapping each agent's ID (usually a string) to its corresponding action (line 6). Similarly, signals such as observations, rewards, terminations, truncations, and additional information are consolidated into dictionaries mapping agent IDs to their respective signals (line 8). Relying on these dictionaries allows providing all information at once.

The second API, termed *agent-environment cycle* (AEC), suits turn-based scenarios, such as most board games [Terry et al., 2021]. A typical usage of this API is depicted in Listing 3. In this setup, each loop turn furnishes information solely for the agent currently taking its turn via a call to the `last()` function (line 7). In some cases, such as when dead or blocked, agents can choose to perform no action by playing `None` (lines 8–9).

Unlike MO-Gymnasium, these environments support a variable number of agents within episodes, and an episode typically terminates when no agent can act. For additional notes on the APIs, refer to the documentation website: https://momaland.farama.org/api/aec/.

```python
from momaland.envs.multiwalker_stability import momultiwalker_stability_v0 as _env

env = _env.env(render_mode="human")
env.reset(seed=42)
for agent in env.agent_iter():
    # vec_reward is a numpy array
    observation, vec_reward, termination, truncation, info = env.last()
    if termination or truncation:
        action = None
    else:
        action = policies[agent](observation)
    env.step(action)
env.close()
```

Listing 3: AEC API usage for turn-based scenarios.

These APIs enable the modeling of all our benchmarking environments and offer the advantage of aligning closely with PettingZoo's conventions, thus facilitating comprehension for MARL practitioners and reuse of existing utilities such as the SuperSuit's wrappers [Terry et al., 2020]. Additionally, MOMAland provides utilities to expose most environments through both APIs (except for some board games, where support for the parallel API is deemed unnecessary).

5.4 Wrappers and Utilities

In addition to environments and standard APIs, MOMAland provides useful utilities tailored to aid algorithm designers in creating solving approaches for these new settings.

Firstly, the library offers wrappers that allow modifying one aspect of the wrapped environment, such as normalizing observation. Importantly, MOMAland environments are compatible with PettingZoo and SuperSuit wrappers, as long as they do not alter the reward vectors. This allows relying on stable implementations and avoiding code duplication. Additionally, MOMAland includes wrappers dedicated to handle vectorial rewards, addressing the distinction from PettingZoo.

For instance, the *NormalizeRewards(idx, agent_id)* wrapper facilitates the normalization of the idx^{th} immediate reward component for a specified agent. Furthermore, the *LinearizeRewards* wrapper enables the transformation of agent reward vectors into scalar values through the usage of a weighted sum scalarization, thereby converting multi-objective environments into single-objective ones under the standard PettingZoo API (Figure 5.1, bottom right arrow). While this adaptation allows for the utilization of existing multi-agent RL algorithms to learn a designated trade-off, it restricts the discovery to points solely on the convex hull of the PF, see Figure 2.9. Secondly, the *CentralizeAgents* wrapper compresses the multi-agent dimension into a single centralized agent, providing direct conversion to the MO-Gymnasium API (Section 4.1), see Figure 5.1 bottom left arrow. This adaptation enables learning using single-agent multi-objective algorithms, such as those featured in MORL-Baselines (Section 4.2).

Finally, MOMAland includes a set of baselines algorithms showing example usage of the API and previously discussed utilities. These baselines are discussed in more detail in the next section.

Algorithm	Reward Utility		Obs. space	Act. space
MOMAPPO: Decomposition + MARL (Section 5.5.1)	Team	Team, Linear	c/d	c/d
Centralization wrapper + MORL algorithm (Section 5.5.2)	Team	Team, Any	d	d

TABLE 5.2: Baselines solving methods available with MOMAland.

Algorithm 5.1 MOMAPPO using decomposition (MOMARL/D)

Input: Number of weight vector candidates n, stopping criterion per weight *stop*, Environment *MOMAenv*.
Output: A Pareto set of multi-agent policies $\mathcal{PS}$.

1: $\mathcal{PS} = \emptyset$
2: $\mathcal{F} = \emptyset$
3: **for** $i \in [1, n]$ **do**
4: $\quad \mathbf{w}$ = GenerateWeights($\mathcal{F}$)
5: $\quad$ *NormEnv* = NormalizeRewards(*MOMAenv*)
6: $\quad$ *MAEnv* = LinearizeRewards(*NormEnv*,$\mathbf{w}$)
7: $\quad \boldsymbol{\pi}$ = MAPPO(*MAEnv*, *stop*)
8: $\quad \mathbf{v}^{\boldsymbol{\pi}}$ = EvaluatePolicy(*MOMAenv*, $\boldsymbol{\pi}$)
9: $\quad$ Add $\boldsymbol{\pi}$ to $\mathcal{PS}$ and $\mathbf{v}^{\boldsymbol{\pi}}$ to $\mathcal{F}$ if $\mathbf{v}^{\boldsymbol{\pi}}$ non-dominated in $\mathcal{F}$
10: **end for**
11: **return** $\mathcal{PS}$

5.5 Baselines Solving Methods

In this section, we aim to demonstrate the applicability of existing techniques coming from MARL and MORL to construct new solving methods for cooperative MOMARL problems. We provide baselines that allow learning under different settings. These baselines are listed in Table 5.2. The second and third columns refer to the classification made in the work of Rădulescu et al. [2020]. We identify two main ways to learn multiple multi-agent policies: transforming the MOMA problem into several single-objective MA problems and using an MARL algorithm (*i.e.*, using decomposition as in MORL/D, Section 3.2), or centralizing all the agents into a single global agent and use an MORL algorithm to solve the single-agent MO problem. One way to see these two alternatives is that the first one compresses the multi-objective dimension, and the second one compresses the multi-agent dimension. It is worth noting that these algorithms do not aim to be the most efficient, but provide a solid foundation for future work. Finally, other learning methods, designed for adversarial or mixed settings with known utility are also presented in the MOMAland paper.

5.5.1 Solving MOMARL Problems Using Decomposition

Algorithm 5.1 describes the extension of the MAPPO algorithm [Yu et al., 2022] (CTDE and parameter sharing, as explained in Section 2.4) to return a Pareto set of multi-agent policies in cooperative problems. Akin to MORL/D (Section 3.2), it employs the decomposition technique to divide the multi-objective problem into a collection of single-objective problems which are then solvable with a multi-agent RL algorithm. In this context, a scalarization function, parameterized by weight vectors, allows performing the decomposition and targeting various areas of the objective space. The most common scalarization function, weighted sum, is used in this algorithm for its simplicity (through the *LinearizeRewards* wrapper, line 6). Notice that the rewards of

Algorithm 5.2 MOMARL using centralization

Input: Environment *MOMAenv*.
Output: A Pareto set of centralized multi-agent policies $\mathcal{PS}$.

1: $MOEnv$ = CentralizeAgents($MOMAenv$)
2: $\mathcal{PS}, \mathcal{F}$ = MORL($MOEnv$)
3: **return** $\mathcal{PS}$

the environment are first normalized to mitigate the difference in scale of each objective (line 5). Moreover, similar to MORL/D design choices, the weight vectors can be generated in different manners, *e.g.*, optimistic linear support (OLS) [Roijers and Whiteson, 2017], GPI-LS [Alegre et al., 2023], uniformly [Das and Dennis, 2000, Blank et al., 2021], or randomly (line 4). After training a multi-agent policy for a given trade-off using MAPPO [Yu et al., 2022], the policy is evaluated on the original environment, allowing to compute an estimate of $\mathbf{v}^{\boldsymbol{\pi}}$ (line 8) and add the policy to the Pareto archive of multi-agent policies if it is non-dominated (line 9). Finally, the algorithm returns all non-dominated multi-agent policies (line 11).

It is worth noting that this algorithm implementation is a straightforward combination of MARL and MORL techniques. We believe it can be improved by adding non-linear scalarization, advanced weight vector generation method, or cooperation between single-objective subproblems such as conditioned networks [Abels et al., 2019] or transfer [Natarajan and Tadepalli, 2005]. A thorough review of such techniques in the context of MORL was held in Sections 2.3.4 and 3.2.

5.5.2 Solving MOMARL Problems Using Centralization

Algorithm 5.2 describes the process of training a MOMARL agent using the centralization wrapper introduced earlier. This technique condenses the multi-agent dimension and trains a central policy conditioned on a global state, which generates joint actions. In this context, MORL learning algorithms from MORL-Baselines are employed.

However, akin to single-objective MARL, this approach has limited applicability due to the action space becoming unwieldy with an increase in the number of agents [Schroeder de Witt, 2021]. Additionally, deploying centralized policies in decentralized settings poses challenges, restricting the application of such techniques to scenarios with robust communication and susceptible to single points of failure. Despite their infrequent usage in practice, these methods often serve as baselines for research in MARL. Moreover, these provide a nice bridge to the methods explored in the first part of the thesis. Hence, we believe it is a relevant addition to our library.

5.6 Experiments

This section shows the results of our two approaches (decomposition and centralization) to solve collaborative MOMARL problems included in MOMAland.

Metrics. Similar to our works in single-agent MORL, we use various performance indicators as presented in Section 2.2.3: hypervolume, expected utility, cardinality, and sparsity. Each set of experiments has been run on 10 different seeds for statistical robustness.

	Hyperparameter	Value
MAPPO	Actor hidden layers	[256; 256]
	Critic hidden layers	[256; 256]
	Activation	tanh
	Anneal learning rate	true
	Clip epsilon	0.2
	Entropy coefficient	0.
	GAE lambda	0.99
	Learning rate	0.001
	Max grad norm	0.5
	Number of mini-batches	2
	Number of steps per epoch	1280
	Updates per epoch	2
	VF coefficient	0.8
MOMAPPO	Timesteps per weight	500,000
	Num weights	20
	Weight generation	Uniform [Blank et al., 2021]

TABLE 5.3: Hyperparameter values used for MOMAPPO in our experiments.

Environments. To validate our approaches, we use different environments implemented in MOMAland. MOMAPPO (Algorithm 5.1) is validated on *mo_multiwalker_stability_v9* (Figure 5.2, third picture), an environment with continuous actions and observations. This environment is a MODec-POMDP [Rădulescu et al., 2020] where multiple agents (the walkers) are tasked to carry a package to the right side of the screen without any walker or package falling. The rewards involve going to the right side of the screen to make progress as well as avoiding shaking the package. Notably, going faster usually implies making more brutal moves and thus shaking the package, making these objectives contradictory.

Our centralization-based approach is validated on *moitem_gathering_v0* (Figure 5.2, rightmost picture). This environment, adapted from Källström and Heintz [2019], is a multi-agent grid world containing items of different colors. Each color represents a different objective (by default 3, but configurable) and the goal of the agents is to collect as many objects as possible. This environment supports discrete observations and actions. This allows using our centralization wrapper as well as MORL-Baselines algorithms.

Implementations details. Hyperparameters values used to perform these experiments are reported in Table 5.3 for MOMAPPO, and we used the default hyperparameters for MORL-Baselines' algorithms. The performance indicators plotted are the average and 95% confidence interval reflecting the general performance of the algorithm over random seeds ranging between 0 and 9 included, whereas the PF plot gives an idea of the final result for a given run. The reference point used for hypervolume calculation was $[-300, -300]$ for *mo_multiwalker_stability_v9*, and $[0, 0, 0]$ for *moitem_gather_v0*. Our experiments have been performed on the high-performance computers of the University of Luxembourg [Varrette et al., 2014] and of the Vrije Universitet Brussels. Raw data of the training processes can be found in Open RL Benchmark [Huang et al., 2024], and on `https://wandb.ai/rradules/MOMAland-IG-3?nw=nwuserflorianfelten`.

MOMAPPO results. Figure 5.3 illustrates the MO metrics results that have been obtained by running MOMAPPO (Algorithm 5.1) on *mo_multiwalker_stability_v9*. A first observation to note is that this technique is able to learn diverse trade-offs as the MO metrics improve over time. Notably, the usage of a Pareto archive allows having a monotonically improving PF over the training process. Indeed, all indicators seem to increase over the training course. Despite using 20 different weights, only 3 Pareto optimal policies are kept on average. This outcome arises from the approach of training a policy from scratch for each new weight. There is potential

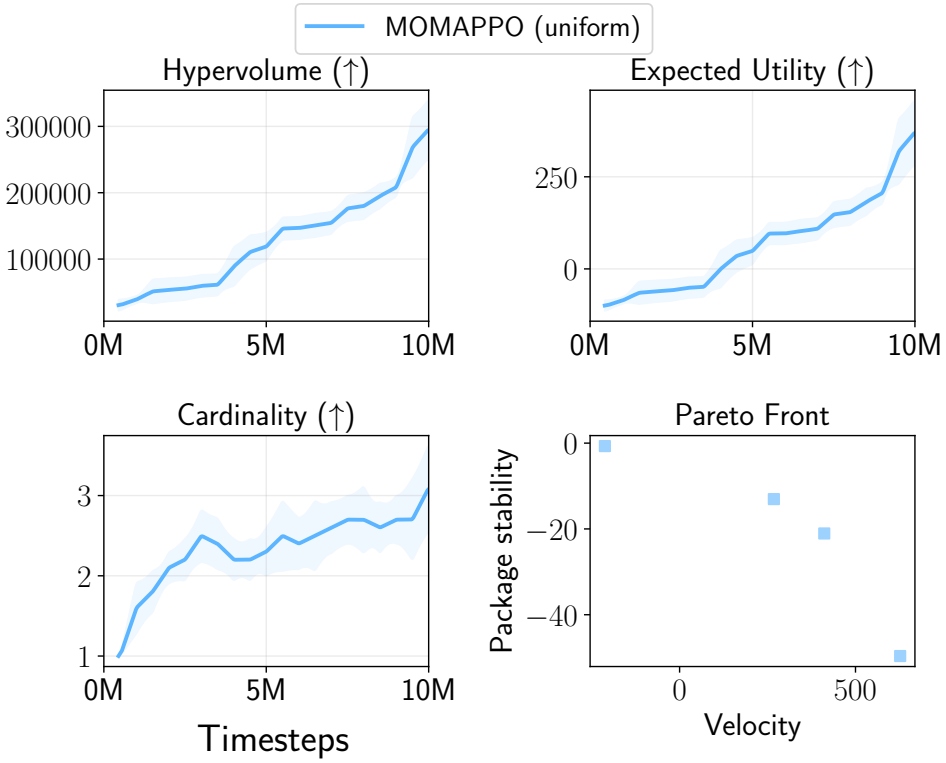


FIGURE 5.3: Average and 95% confidence intervals of multi-objective performance indicators on training results from MOMAPPO with 20 uniform weights on *mo_multiwalker_stability_v9*. The Pareto Front plot has been extracted from the run with the highest hypervolume.

for improvement by employing cooperation methods as in MORL/D, such as transfer learning, to enhance the performance of policies across different weights.

Centralization results. Figure 5.4 illustrates the results obtained via the centralization method and using PCN [Reymond et al., 2022] and GPI-LS [Alegre et al., 2023] to solve the MORL problem. Here also, both methods seem to improve the PF learned over the course of the training process. Similar to single-agent MORL settings, GPI-LS exhibits strong performance in this domain.

These conclusive results in the initial experiments show the transferability of methods across domains and pave the way for future algorithmic improvements. As a final note, we would like to discuss the limitations of the current techniques. Centralization, as observed in MARL, faces challenges in scalability due to the explosion of action space as the number of agents increases. Conversely, decomposition struggles to scale effectively with the growing number of objectives, as the required number of weight vectors for comprehensive coverage of the objective space grows exponentially. Therefore, there is a pressing need for the development of algorithms tailored to scenarios involving numerous agents and objectives.

While MOMAland supports settings such as adversarial environments, these go beyond the scope of this thesis and are thus not discussed in this manuscript. For the interested reader, the paper introducing MOMAland [Felten et al., 2024b] presents solving methods for adversarial settings, too. Importantly, there is currently no existing solution concept for adversarial or mixed settings with unknown preferences, this means that certain environments introduced in MOMAland require answering open scientific questions.

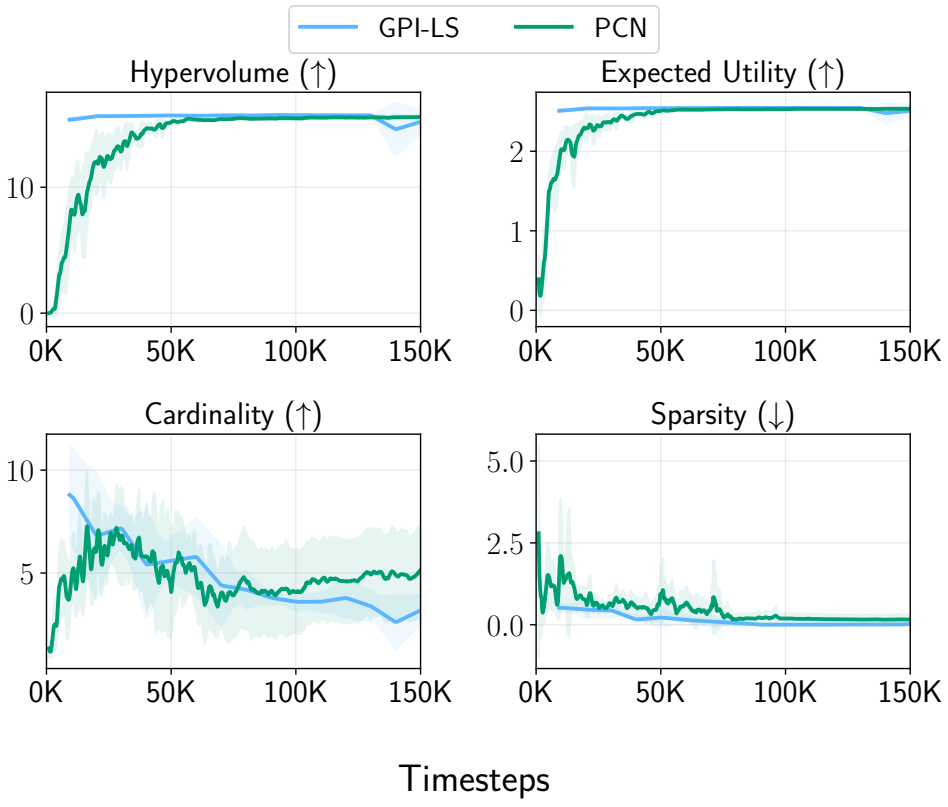


FIGURE 5.4: Average and 95% confidence interval of MO performance indicators on training results from single-agent MORL algorithms and centralization wrapper on *moitem_gathering_v0*.

5.7 Summary

In this section, we presented MOMAland, a novel library aimed at facilitating MOMARL research. MOMAland contains standard APIs and a collection of environments dedicated to establishing solid foundations for the nascent field of MOMARL. Moreover, MOMAland includes utilities and learning algorithms serving as baselines for evaluating novel approaches. As an open-source platform open to contributions, MOMAland is positioned to evolve in tandem with this new research field, fostering collaboration and providing a shared platform for researchers to explore and innovate. Building upon these advancements, we present real-world applications leveraging MOMARL techniques in the subsequent chapter.

Published Work

The contributions described in this part are currently under review, a pre-print version of the article is available on arXiv:

- Florian Felten, Umut Ucak, Hicham Azmani, Gao Peng, Willem Röpke, Hendrik Baier, Patrick Mannion, Diederik M. Roijers, Jordan K. Terry, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, and Roxana Rădulescu. MOMAland: A Set of Benchmarks for Multi-Objective Multi-Agent Reinforcement Learning. 2024b

Chapter 6

Application: Learning Multi-Objective Swarming Formations with Drones

Contents

As mentioned earlier, RL has garnered significant research attention in recent years, driven in part by its impressive successes in challenging applications such as Go [Silver et al., 2016] and drone racing [Kaufmann et al., 2023]. This field has expanded into more demanding scenarios, including training multiple agents in MARL, training a single agent to learn to make compromises between multiple conflicting objectives in MORL (Section 2.3), or combining both extensions in MOMARL (Section 2.4) [Rădulescu et al., 2020]. These extensions have enhanced the modeling capabilities of RL, broadening its applicability to new problem domains [Bayındır, 2016, Hayes et al., 2022, Lu et al., 2022b, Hu et al., 2023].

Among these applications, RL has been employed to autonomously generate behaviors for controlling robot swarms [Bayındır, 2016, Hüttenrauch et al., 2019]. In such settings, agents typically learn to cooperate to accomplish tasks that would be unattainable individually, such as search and rescue missions. Particularly, swarm formation and learning to capture an evader, phenomena commonly observed in nature, have been identified as challenging applications in robotics [Birattari et al., 2019, Hüttenrauch et al., 2019]. Moreover, the application of such learned behaviors to real-world robots has been facilitated by the recent introduction of cheap reliable robots such as the Crazyflie nanodrones [Giernacki et al., 2017].

While these problems have traditionally been modeled as single-objective (SO) MARL problems [Bayındır, 2016], many real-life problems typically consist in making compromises between multiple objectives. For instance, achieving stable formation around an object or pursuing an evader presents intriguing MOMARL problems: these scenarios require making trade-offs between multiple objectives for multiple collaborating agents, as the agents aim to minimize their distance to the object/evader while maximizing their distance from each other to prevent formation collapse. However, to the best of our knowledge, these problems have never been tackled using a fully multi-objective multi-agent approach. Hence, this part of the thesis aims to provide solutions for such MOMARL problems by relying on the accumulated knowledge previously presented in the thesis.

Before diving into the contribution, it is worth remembering that earlier chapters of the thesis introduced various methods for solving MORL and MOMARL problems, demonstrating improved sample efficiency compared to naive baselines. However, a new aspect arises in our application setting: the simulated environment can be implemented and executed on graphical processing units (GPUs). This enables significant improvements in terms of sample throughput, introducing novel considerations regarding the development of MO(MA)RL algorithms, which are explored in this chapter.

6.1 CrazyRL: Accelerated MOMARL for CrazyFlie Swarms

RL and its extensions are known to demand a substantial number of samples from the environment to train agents effectively. Consequently, a considerable amount of RL research has been dedicated to improving training efficiency. There are two common ways to improve RL algorithms: (1) increase the amount of information learned for each sample taken from the environment (sample efficiency), and (2) increase the amount of sample per time unit the algorithm can extract from the environment (sample throughput). Up to now, a significant amount of (MO)RL research has been focused on sample efficiency by enhancing existing algorithms through the usage of complex techniques, with sample throughput being seen more as an engineering hurdle rather than a subject of active research. However, recent software developments have greatly simplified the burden of increasing the sample throughput of RL algorithms. An example of such software is JAX [Bradbury et al., 2018], which enables easy use of multiple cores on hardware-accelerated devices like graphical processing units (GPUs) or tensor processing units (TPUs) through a high-level NumPy-like API [Harris et al., 2020]. RL algorithms and environments implemented solely in JAX have demonstrated performance improvements of up to 4000 times compared to traditional RL setups, where the agent's learning process occurs on an accelerator while the environment(s) run on the CPU [Lu et al., 2022a]. These speed enhancements have led to a resurgence of simpler but faster techniques being applied to problems, rather than relying on more complex and sample-efficient approaches.

In this part, we aim to demonstrate that the multi-objective extensions of RL can also benefit from such accelerations by directly implementing MOMARL environments and agents on accelerators. Furthermore, we aim to showcase the complete process of deploying MOMARL techniques in real-world contexts. Our contributions can be summarized as follows:

- In Section 6.2, we introduce various MOMARL problems which consist of drones learning formation despite conflicting objectives. These are implemented both on CPU and accelerated devices;
- In Section 6.3, we propose MOMAAD, a novel **M**ulti-**O**bjective **M**ulti-**A**gent RL algorithm based on hardware **A**cceleration and **D**ecomposition. It combines MARL algorithms with "embarrassingly parallel computing" [Wilkinson and Allen, 2005, Régin et al., 2013] to efficiently learn a PS of multi-agent policies. We also refer to this new techniques as "*accelerated decomposition*";

- In Section 6.4, we demonstrate that MOMAAD can learn various trade-offs on the introduced environments within seconds, and this process scales sublinearly with the desired number of compromises. Compared to state-of-the-art methods which imply running environments on the CPU, our method reaches speedups of order superior to 2000 on consumer-grade computing devices;
- In Section 6.5, we validate the effectiveness of our approach by conducting experiments with real drones, affirming its utility for swiftly prototyping simulation-to-reality behaviors.

6.2 Environments

As outlined in Section 2.4.1, our approach to modeling multi-objective swarm formation problems relies on the MOMMDP framework. The CrazyRL (short name for RL with Crazyflie drones) environments, included in MOMAland, are 3 MOMMDPs – Surround, Escort, and Catch – designed to facilitate the learning of high-level swarm formations around potentially moving objects for multiple drones. These environments rely on high-level control commands, such as a 3D speed vector, indicating where each drone should go, rather than low-level control like torque in the engine. This choice significantly simplifies the state and action spaces of the agents, enabling them to focus on the core problem of learning formation and eliminating the need for heavier robotics simulators such as Gazebo [Koenig and Howard, 2004] or Pybullet [Coumans and Bai, 2016, Tai et al., 2023].

In practice, each agent (drone) perceives its current x, y, and z coordinates (denoted as x_i, y_i, z_i for agent i) along with the target coordinates (x_{targ}, y_{targ}, z_{targ}). Additionally, agents also perceive the positions of other agents, making the environment fully observable. At each time step, agents select a 3D speed vector as their action, dictating the direction in which they wish to move, *i.e.*, $a_i \in [-1, 1]^3$, $\forall i \in [1, n]$. The drones' movements are discrete, with their positions updated at each step by applying these action vectors, effectively "teleporting" them. If the moves lead outside coordinates specified by the map size, the new coordinates of the agents are clipped to stay inside the map. The global state, used for CTDE learning, is a concatenation of all known positions (agents and target). Episodes terminate upon drone collisions, contact with the floor or target, or when a predefined number of time steps is reached.

In the three specified environments, the reward function contains two conflicting objectives: (1) minimizing the distance to the shared target while simultaneously (2) maximizing the distance from the other agents. In multi-objective settings, the goal is generally to maximize all objectives, requiring the need to transform the minimizing objective into a maximization of its negation. However, we noticed that transforming the first reward component into a negative value and maximizing both components can adversely affect learning performance on the studied policy optimization algorithm (PPO) [Schulman et al., 2017]. Therefore, we opted to convert the first objective into a potential-based reward instead [Ng et al., 1999].

For each agent i, the rewards are formally defined as follows:

$$\begin{aligned} r_{1,i}(\mathbf{s}, \mathbf{a}) = &\|(x_i^{t-1}, y_i^{t-1}, z_i^{t-1}) - (x_{\text{targ}}^{t-1}, y_{\text{targ}}^{t-1}, z_{\text{targ}}^{t-1})\|^2 \\ &- \|(x_i^{t}, y_i^{t}, z_i^{t}) - (x_{\text{targ}}^{t-1}, y_{\text{targ}}^{t-1}, z_{\text{targ}}^{t-1})\|^2, \end{aligned}$$

$$r_{2,i}(\mathbf{s}, \mathbf{a}) = \frac{\sum_{j \neq i} \|(x_i^t, y_i^t, z_i^t) - (x_j^t, y_j^t, z_j^t)\|^2}{n-1},$$

where $r_{o,i}(\mathbf{s}, \mathbf{a})$ is the o^{th} objective value of agent i, and x_i^t denotes the x position of agent i at time step t. These individual rewards are then aggregated to form a multi-objective team reward: $\mathbf{r}(\mathbf{s}, \mathbf{a}) = \sum_{i \in [1,n]} \mathbf{r}_i(\mathbf{s}, \mathbf{a})$.

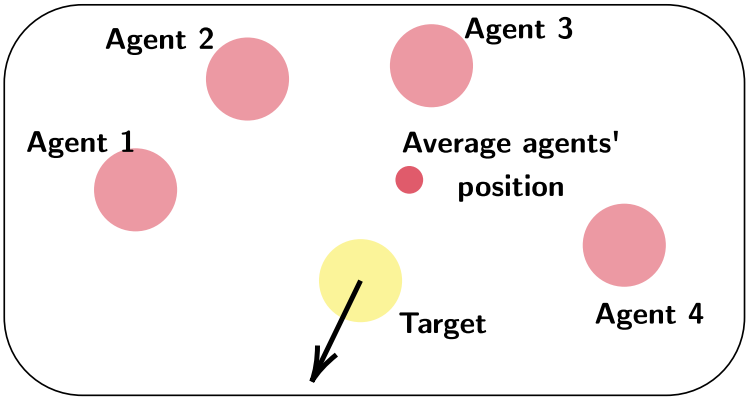


FIGURE 6.1: Target move decision in the Catch environment (flattened to 2 dimensions for illustrative purpose).

The following paragraphs describe the differences between the three environments. Note that a visualization of our simulator is available in Figure 5.2, second picture.

Surround. In this environment, the goal is for the drones to establish a stable formation around a fixed target.

Escort. This environment is an extension of the previous one and introduces the added challenge of a moving target. In this scenario, the target is assigned an initial position and a final position, and it moves linearly from the former to the latter in a specified number of time steps.

Catch. In this last environment, an element of intelligence is introduced into the target's behavior. Specifically, the target tries to escape the agents by calculating an average position of these and endeavors to move in the opposite direction. An example of this strategy in 2 dimensions is exposed in Figure 6.1. However, if the computed average position is too close to the current target position, the target resorts to random movement as a strategy.

Finally, an intriguing aspect of these environments is they have simple dynamics, enabling direct implementation on accelerated hardware. The next section elaborates on this new setting.

6.3 MOMARL in Fast Simulation Regime

There are two ways to improve the performance of RL algorithms in general: improve sample efficiency, or improve sample throughput. While numerous innovations have concentrated on the former, involving the introduction of new techniques to optimize the utilization of each collected sample from the environment, less attention has been given to the latter. There are multiple reasons why sample efficiency prevailed over sample throughput as a research topic.

First, in RL research, it is customary to evaluate agent performance throughout the training process in terms of collected rewards over samples collected from the environment, usually represented as a learning curve. Hence, introducing a new RL algorithm typically involves comparing it to existing baselines in terms of sample efficiency on benchmark environments. However, sample-efficient algorithms often involve complex operations, resulting in slower sample throughput compared to simpler algorithms. Unfortunately, this fact is seldom included in research papers. While sample efficiency can correlate with time efficiency, it is possible that a simpler yet faster

algorithm combined with a fast environment could reduce the total time needed to reach acceptable performance, thus yielding better results for end users, *e.g.*, see the GPI-LS vs. PCN discussion in Section 4.3. To address this bias towards sample efficiency, some authors have also started reporting time efficiency, evaluating agents' performance against wall time [Huang et al., 2024].

Moreover, improving sample throughput traditionally required a substantial engineering effort for accelerating environments. Indeed, achieving faster RL implementations often involved delving into low-level programming languages like C++ or addressing challenges related to concurrency and distributed programming in order to harness additional computational cores. Recently, a new set of programming tools designed to enhance the speed of machine learning methods, exemplified by JAX [Bradbury et al., 2018], has emerged. JAX serves as a numerical computing library for machine learning, offering a Python API and supporting standard functionalities like automatic differentiation and the execution of linear algebra operations on high-performance hardware such as GPUs or TPUs. Furthermore, JAX facilitates the compilation of user-defined Python functions into low-level kernels, thereby eliminating the reliance on Python control flow during the execution of these kernels. Additionally, JAX enables Single Instruction, Multiple Data (SIMD) [Flynn, 1966] programming through its `vmap` and `pmap` instructions. `vmap` takes a function as parameter and generates a new function that takes input and produces output with an extra dimension, enabling the vectorization of the function on hardware accelerators and thus using multiple cores on the accelerator. `pmap` parallelizes function computation across multiple devices, enabling the usage of even more cores. Leveraging this novel programming abstraction, several RL projects have recently been introduced. For instance, Hessel et al. [2021] presents multiple architectures that harness the computational power of accelerators. The *Sebulba* architecture distributes the learning component of the RL process across multiple accelerators using the `pmap` operator, while the *Anakin* architecture co-locates the environment and learning processes on the accelerator. While this latter approach necessitates a full JAX implementation of the environment, it delivers substantial enhancements in terms of sample throughput: (a) environments can be effortlessly vectorized (using `vmap` on their exposed functions), (b) eliminating the need for costly data transfers between the CPU and the accelerator, and (c) the complete agent-environment interactions can be compiled into a single kernel, facilitating operation fusion, obviating the need for Python control flow, and allowing to optimize computations. The *Anakin* architecture has been recently implemented for single-objective RL in PureJaxRL [Lu et al., 2022a], relying on JAX-implemented environments such as Brax [Freeman et al., 2021], Gymnax [Lange, 2022], or Pgx [Koyamada et al., 2023], and has demonstrated speedups on the order of 4000 compared to RL implementations relying on CPU-based environments. Taking a step further, the authors of PureJaxRL also proposed training RL agents in an embarrassingly parallel manner [Wilkinson and Allen, 2005, Régin et al., 2013], enabling the parallelization or vectorization of training across different seeds or hyperparameters. This was trivially achieved by using the SIMD primitives of JAX on the training function.

6.3.1 Accelerated Decomposition

Thanks to the natural independence between subproblems offered by the decomposition scheme (MORL/D, MOMARL/D), our method maximizes sample throughput by solving the MO problem in an embarrassingly parallel manner [Wilkinson and Allen, 2005, Régin et al., 2013]: solving the SO problems independently, in parallel, using simple yet fast methods (MAPPO in our case). While the concept of combining embarrassingly parallel computing with decomposition is not novel, previous parallelization efforts predominantly relied on multiple CPU threads across one or more devices. What sets our approach apart is that we fully harness the potential of accelerators instead. This achievement allows us to scale up our algorithm to utilize thousands of computing cores as well as removes costly memory transfers to perform accelerated decomposition in order to

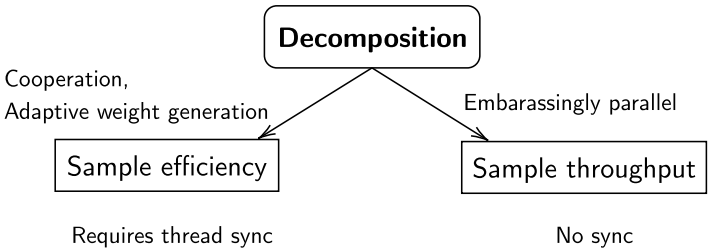


FIGURE 6.2: Solving paradigms for decomposition algorithms: sample efficiency vs. sample throughput. MORL/D (Section 3.2) takes the left approach, while this work takes the right approach.

learn a Pareto set of policies. Our idea builds upon the idea of PureJaxRL which trained for multiple seeds at once by using `vmap` over the training function, except here we train over weight candidates.

Such an approach aligns with observations made in Sutton's "*Bitter lesson*," which posits that: "*The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin.*" In a similar vein, adhering to Amdahl's law [Amdahl, 1967], it becomes apparent that there is a tension between efforts to enhance techniques such as using cooperation mechanisms between SO subproblems or adaptive weight vector generation (the sample efficient way, such as in MORL/D) and allowing programs to run in parallel, employing a more brute-force approach (the sample throughput way), see Figure 6.2.

A pseudo-code of our MOMARL algorithm which relies on accelerated decomposition (MOMAAD) is listed in Algorithm 6.1. The procedure starts by generating a set of weight candidates in the objective space (line 4). Then, for each of these candidates, the MO environment is turned into a SO environment using a wrapper (similar to MO-Gymnasium or MOMAland's, Sections 4.1 and 5.4) which applies a scalarization function (weighted sum in our case) to the reward vector, and a multi-agent policy is learned using standard MARL procedure (with parameter sharing to limit the number of parameters to learn, see Section 2.4) (lines 7–11). This step is

Algorithm 6.1 MOMAAD: **m**ulti-**o**bjective **m**ulti-**a**gent reinforcement learning based on **a**ccelerated **d**ecomposition

1: **Input:** Number of weight candidates c, Multi-Objective Multi-Agent environment *MOMAEnv*
2: **Output:** A Pareto set of policies $\mathcal{PS}$ and its linked front $\mathcal{F}$
3:
4: $\mathcal{W}$ = Generate c weight vectors
5: Policies = $\emptyset$
6: **compile and run** {
7: **for** $\mathbf{w} \in \mathcal{W}$ **do** ▷ In parallel using vmap or pmap
8: *MAEnv* = ScalarizeRewards(*MOMAEnv*, $\mathbf{w}$)
9: $\boldsymbol{\pi}$ = MARL(*MAEnv*)
10: Add $\boldsymbol{\pi}$ to Policies
11: **end for**
12: }
13:
14: Evaluations = $\emptyset$
15: **for** $\boldsymbol{\pi} \in$ Policies **do**
16: $\mathbf{v}^{\boldsymbol{\pi}}$ = EvaluatePolicy(*MOMAEnv*, $\boldsymbol{\pi}$)
17: Add $\mathbf{v}^{\boldsymbol{\pi}}$ to Evaluations
18: **end for**
19: $\mathcal{PS}, \mathcal{F}$ = ParetoPrune(Policies, Evaluations)
20: **return** $\mathcal{PS}, \mathcal{F}$

effectively vectorized or parallelized by applying one of the SIMD operators from JAX. In practice, it requires changing less than 10 lines of code to turn a fully accelerated MARL algorithm into an embarrassingly parallel MOMARL algorithm using our approach (see Appendix C). The parallel search is fully compiled as a low-level kernel to optimize it for the accelerator (line 6). After all policies have been learned, they are evaluated on the MOMA environment in order to gather their corresponding vector values, $\mathbf{v}^{\boldsymbol{\pi}}$ (lines 14–18). These values are finally used to prune the Pareto-dominated policies and form a Pareto set of policies and Pareto frontier using Equation 2.15 (line 19). Finally, the Pareto set and PF are returned to the user (line 20).

A final note worth mentioning on this algorithm is its similarity with MOMAPPO presented earlier (Algorithm 5.1). This is due to the fact that they both rely on decomposition. However, MOMAAD trains policies in parallel while using a batch of weight candidates whereas MOMAPPO sequentially trains policies and generates new weight vectors, potentially based on the current state of the PF. This way, MOMAPPO allows for trivial transfer of techniques coming from MORL/D (Algorithm 3.6) whereas MOMAAD requires keeping the batches large enough to benefit from parallelization.

6.4 Experiments

In this section, we share the results generated through the usage of MOMAAD on our accelerated environments. First, we outline the experimental configuration and implementation details. Following that, we draw a comparison between fully accelerated MARL and the CPU-based approach within one of our environments for a forced trade-off. Subsequently, we demonstrate the remarkable speed at which our approach enables the learning of multiple compromises in MOMARL environments, with execution times measured in seconds for millions of timesteps, and scaling sublinearly with the number of compromises. Lastly, we test the learned policies by applying them to actual drones, validating their real-world applicability.

6.4.1 Implementation Details

Our library is composed of three environments, implemented in both Python/NumPy following the MOMAland parallel API (Section 5.3), that we call CPU environments, and in pure JAX that we refer to as GPU environments. The code is available at `https://github.com/ffelten/CrazyRL`.

While the JAX-based implementations could not be defined under the MOMAland API due to the differing programming paradigms (functional vs. object-oriented), we tested both variants to ensure their logic and outputs were equivalent. This enables agents to be trained in the JAX-based environments and then execute the resulting policies on the CPU environments. This flexibility facilitates the implementation of visualizations or the control of actual drones, tasks that remain challenging using pure JAX implementations.

Regarding the learning algorithms, our implementation consists of a version of the MAPPO algorithm [Yu et al., 2022] in JAX inspired by PureJaxRL's PPO implementation [Lu et al., 2022a], a fully compiled JAX implementation of MAPPO capable of using vectorized JAX environments, and a JAX implementation of MOMAAD, which builds upon the latter. We chose to implement MAPPO, which is a CTDE method based on PPO [Schulman et al., 2017] because this algorithm is known to be fast, as opposed to more sample efficient, yet slower, methods such as SAC [Haarnoja et al., 2018]. It is worth mentioning that, in our accelerated implementations, the environments and learning algorithms are fully implemented in JAX. This allows the MOMARL loop (Figure 2.16) to be fully compiled and executed onto accelerated devices. This means that no Python control flow is ever called during the learning process.

Name	Value
Hidden layers	[256, 256]
Activation function	tanh
Update epochs per batch	2
Num mini-batches per batch	2
γ	0.99
Learning rate	1e-3
GAE lambda	0.99
Clip epsilon	0.2
Entropy coefficient	0
Value loss coefficient	0.8
Max norm of the gradients	0.5
Learning rate annealing	Yes

TABLE 6.1: MAPPO hyperparameters used in our experiments.

6.4.2 Experimental Setup

To highlight the hardware efficiency of our approach and underscore that RL does not necessarily require the usage of a high-performance computer, we conducted all our experiments on a consumer-grade computer that is 5 years old. This desktop computer, a Dell Aurora model, features an Intel i7-8700 CPU operating at 3.2GHz, accompanied by an NVidia GeForce GTX 1080Ti GPU.

For statistical robustness, we repeated experiments across ten different seeds, ranging from 0 to 9. Most of our experiments employed a consistent set of hyperparameters that were determined through parallel grid search (using JAX's `vmap` on different hyperparameter values) on the studied environments. These hyperparameters are documented in Table 6.1. The only variable hyperparameters changed during our experiments were related to batch creation, influenced by whether we used multiple vectorized environments or a single environment. Specifically, in the case of multiple vectorized environments, the number of steps per batch differed from that in single-environment scenarios. Lastly, all figures presenting execution time incorporate the compilation time of JAX as well. More details related to reproducibility are available in Appendix C.

6.4.3 Fully Accelerated MARL

In this section, our primary objectives are twofold: firstly, to validate the equivalence of our implementations of CPU-based MARL and fully accelerated MARL in terms of sample efficiency, and secondly, to highlight the remarkable sample throughput of the accelerated variants compared to their CPU-based counterparts. The results presented in this section maintain a consistent batch size of 1280 samples across all cases, with adjustments made to the number of steps per batch in accordance with the number of vectorized environments. For example, when employing 10 vectorized environments, the number of steps per batch was set to 128.

Figure 6.3 presents the training results of both our MAPPO implementations on the Surround environment with 3 agents for an hard-coded trade-off, utilizing both the CPU and JAX-based environments. The left part of the figure displays the conventional measure of sample efficiency, revealing that the CPU and GPU versions exhibit similar learning curves when trained on a single environment. However, the true advantage of the GPU version becomes evident when examining wall-time, as it reduces the time required to complete 100,000 timesteps by a few orders of magnitude, as shown in the right part of the Figure and Table 6.2. This aligns with findings reported in PureJaxRL [Lu et al., 2022a].

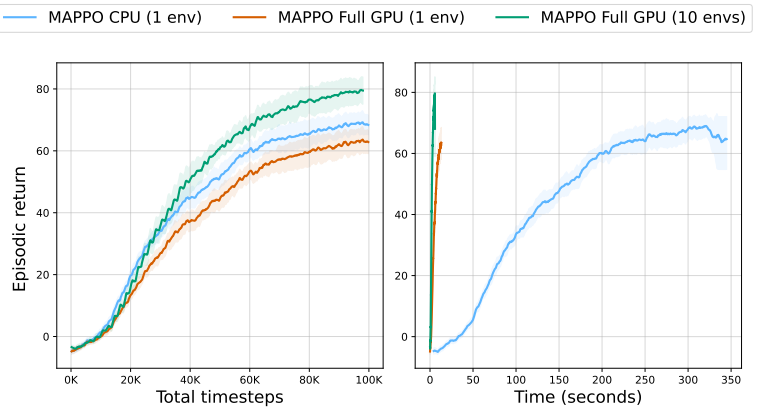


FIGURE 6.3: Mean episodic team return and 95% confidence interval for the Surround environment (for an hard-coded trade-off using MAPPO). Left: sample efficiency, right: wall-time efficiency.

	CPU (1 env)	*GPU (1 env)*	*GPU (10 envs)*
Time	330.7 ± 6.3	12.9 ± 0.04	5.8 ± 0.04
SPS	312.9 ± 31.2	7748.8 ± 24.4	$17{,}306.7 \pm 107.5$
Speedup	-	$\approx$**25**$\times$	$\approx$**55**$\times$

TABLE 6.2: Mean and standard deviation of time (seconds) to perform 100,000 steps and samples per second (SPS) for different implementations of MAPPO.

Furthermore, the obtained results demonstrate that the fully accelerated implementation can readily handle vectorized environments, further enhancing the sample throughput of the method. It is worth noting that with modern accelerators, it is possible to scale the number of environments to thousands, given the abundance of available cores and memory. Observant readers may notice that sample efficiency appears even better in this case compared to non-vectorized environments. We attribute this phenomenon to PPO's documented improved performance with vectorized environments [Huang et al., 2022a]. Additionally, the deterministic nature of the studied environment and the differences in batch creation may contribute to faster learning when using vectorized environments.

6.4.4 Fully Accelerated MOMARL

With the validation of our GPU-based MAPPO implementations against CPU-based environments successfully completed, we now extend our approach to tackle multi-objective problems, as outlined in Algorithm 6.1. In practice, we relied on some functionalities offered by MORL-Baselines (Section 4.2, [Felten et al., 2023a]) and Pymoo [Blank and Deb, 2020] to execute various multi-objective-related tasks, including the generation of weight candidates ([Blank et al., 2021]) and Pareto pruning. All experiments conducted within GPU-based environments were executed with 128 vectorized environments, each performing 10 steps per batch, resulting in a batch size of 1280. Conversely, experiments conducted within CPU-based environments employed a single environment with 1280 steps per batch.

Figure 6.4 demonstrates the remarkable scalability of our embarrassingly parallel algorithm, as evidenced in experiments conducted on the three MOMA environments. Each policy was trained for 3 million steps with a specific weight candidate. Our algorithm showcases the ability to scale sublinearly (assuming linear scaling when looping over MAPPO runs with different weight vectors) with the number of policies being simultaneously learned. This results in an even more dramatic speedup when compared to standard CPU-based environments,

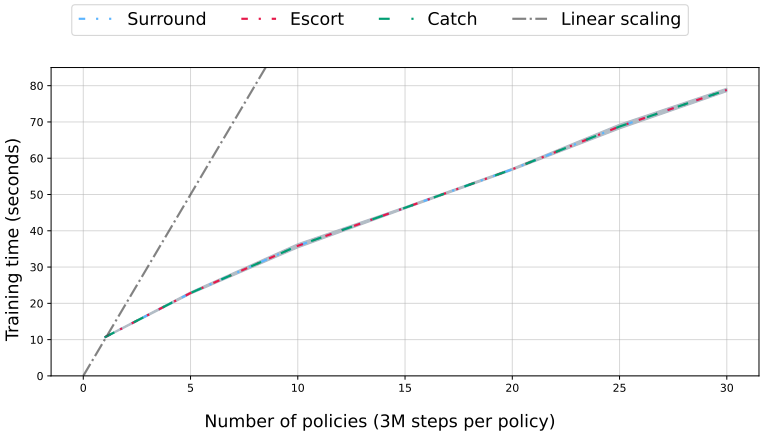


FIGURE 6.4: Mean and 95% confidence interval of time taken to train multiple policies using MOMAAD on various MOMA environments.

Number of policies	*1* (CPU)	*1*	*10*	*20*	*30*
Time	7228.6 ±22.8	10.6 ±0.3	35.9 ±0.9	56.9 ±0.4	78.8 ±0.8
SPS	415 ±1.3	282,251 ± 6809	837,251 ± 20,223	1,053,653 ± 7783	1,141,864 ± 10,858
Speedup	-	**≈680×**	**≈2017×**	**≈2539×**	**≈2751×**

TABLE 6.3: Mean and standard deviation of time (seconds) and samples per second (SPS) to train multiple policies on the Surround environment using MOMAAD.

achieving a staggering feat of completing 90 million timesteps in less than a minute and a half on a typical consumer-grade desktop computer. To facilitate comparison, we include a numerical version of these outcomes related to the Surround environment in Table 6.3. The table also includes the training time for a single policy over 3 million steps using a CPU-based environment on the same Surround environment. Furthermore, the table highlights the speedup factors, demonstrating that our approach achieves speeds up to 2751 times faster than the conventional method. These findings align with the figures reported in PureJaxRL [Lu et al., 2022a]. Notice that the usage of more vectorized environments (128) to learn one policy increases the speedups even more compared to our previous experiments reported in Table 6.2.

Figure 6.5 presents the conclusive outcomes achieved by our algorithm used with 30 weights candidates in the Surround environment, involving 8 agents. The PF after pruning is shown at the top, revealing the diverse trade-offs identified for the various weight candidates. Meanwhile, the bottom part displays a screenshot depicting the final positions attained by the agents, represented as red spheres, encircling the target, symbolized by a yellow sphere. This user-friendly interface enables individuals to select their preferred swarm behavior based on its attached weight vector, offering an alternative to the traditional trial-and-error process where weight vectors are adjusted manually. Moreover, our approach leverages the scalability benefits demonstrated in Figure 6.4.

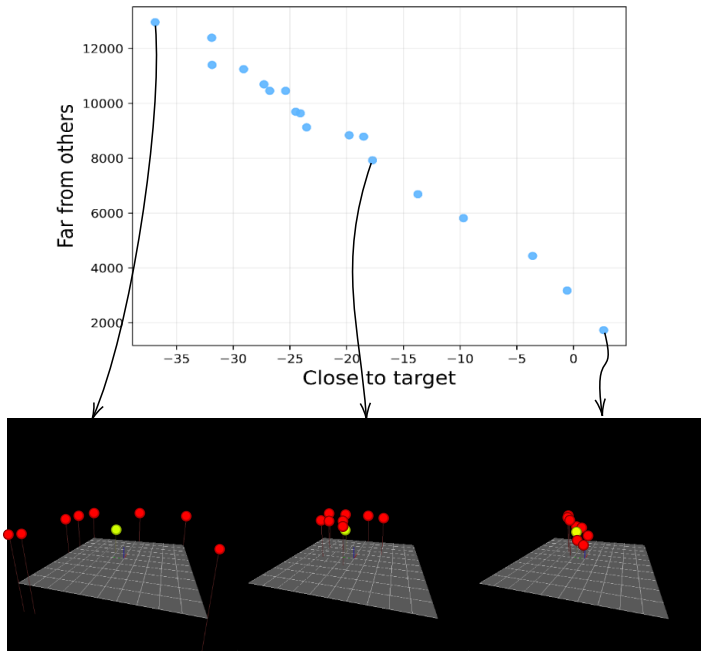


FIGURE 6.5: Pareto front and resulting trade-off policies on the Surround environment.

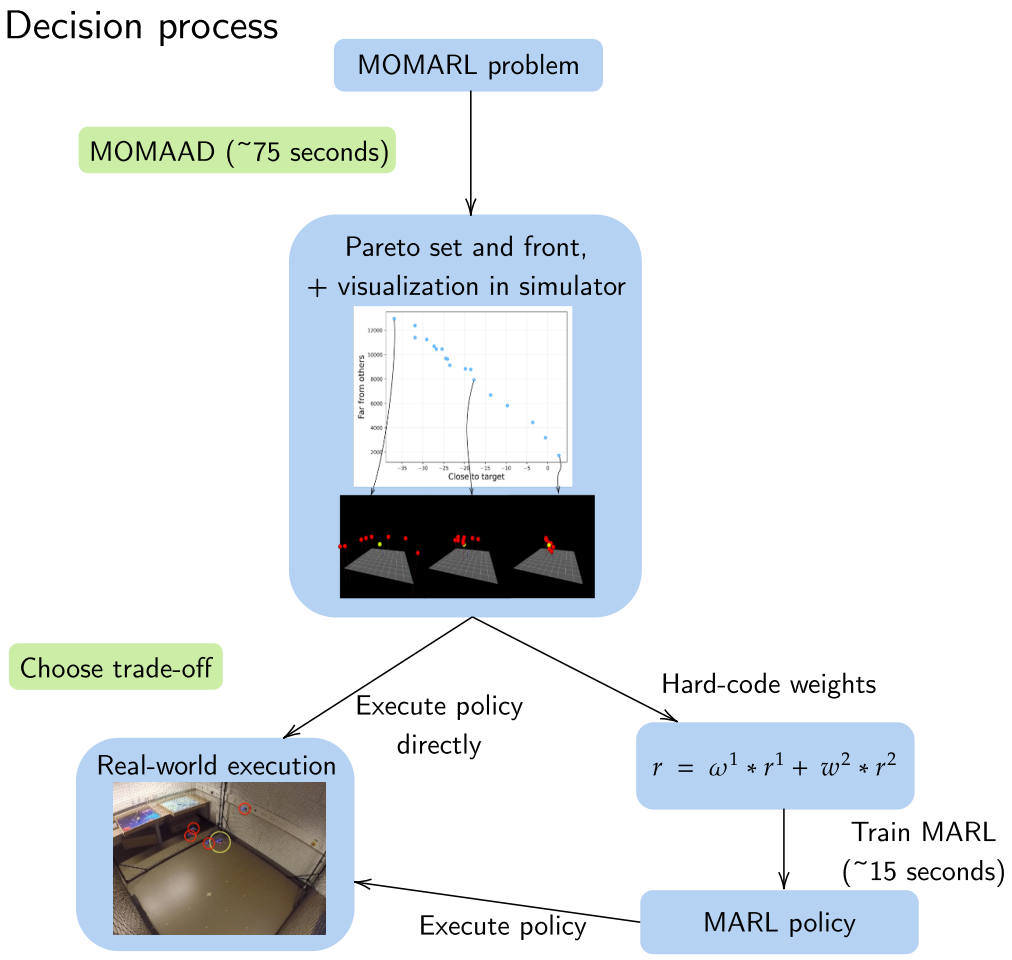


FIGURE 6.6: Decision process when deploying MOMARL agents with CrazyRL.

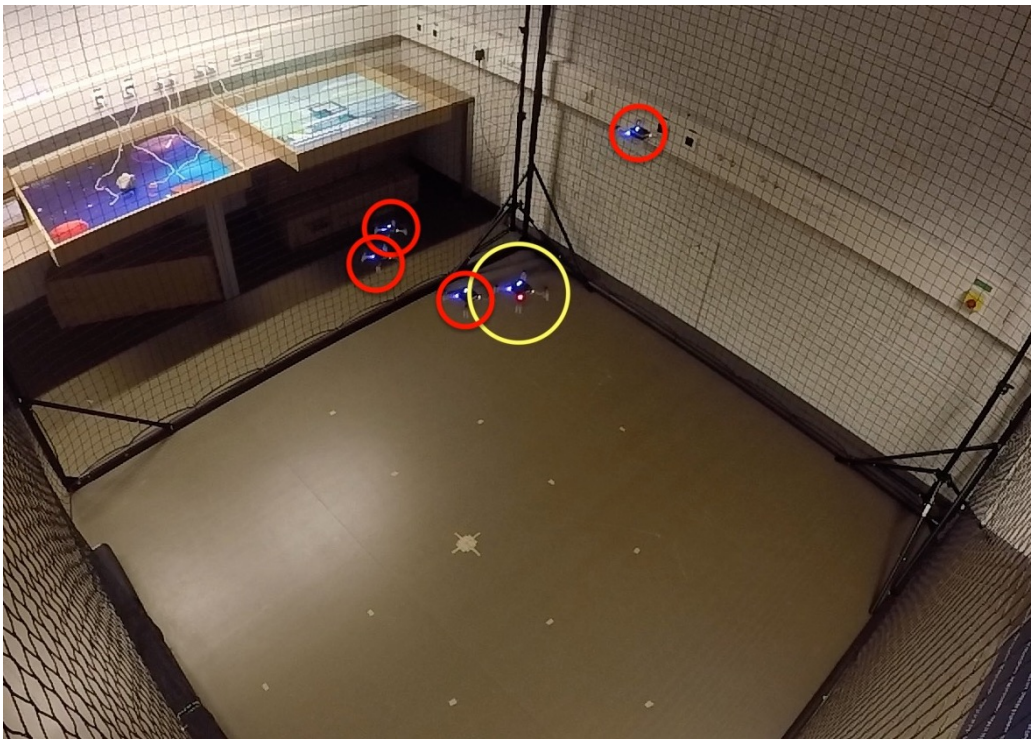


FIGURE 6.7: Execution of one of the learned policies for the Escort environment on real drones.

6.5 Validation in Real Conditions

As previously mentioned, our environments are notably simpler than the typical robotics simulators used for control learning, such as Gazebo [Koenig and Howard, 2004] or Pybullet [Coumans and Bai, 2016, Tai et al., 2023].[1] Nevertheless, to demonstrate the learned formations are still applicable in real-world scenarios, we conducted tests using actual Crazyflie 2 [Giernacki et al., 2017], employing the *high-level commander* provided in the `cflib` by Bitcraze.

The entire decision process used to deploy MOMARL agents is depicted in Figure 6.6. First, it takes about 75 seconds to train 30 policies leading to different trade-offs using MOMAAD. From these policies, it is possible to evaluate them and create a PF, as well as visualize behaviors associated with each trade-off in simulation. This ability to learn behaviors for various weights in an extremely fast fashion (Table 6.3) and then let the user choose contrasts with the traditional expensive trial-and-error approach which requires human input for each trade-off. Based on the presented trade-offs, a user can then determine which behavior and associated weights they want. Next, they can either directly execute one of the learned policies in real-world scenarios, or choose to hard-code its associated weights in code, subsequently reducing the study to a simple MARL problem. In our experiments, training the MARL agents on such scalarized environments took less than 30 seconds in all cases. It should be noted that the hard-coded trade-offs did not generalize to variations in agent numbers or map dimensions in our tests. Specifically, the magnitude of objectives and consequently the behaviors observed for specific weight vectors were altered with changes in the size of the arena or the drone count. This means that it is necessary to recalibrate the weight vector for each new environmental condition, thereby reinforcing the necessity for multi-objective strategies capable of autonomously learning varied trade-offs.

For our real-life experiments, we saved the neural networks' parameters derived from the learned policies in the JAX-based environments and applied them to control the real drones via radio within CPU-based environments.

[1] It is worth mentioning that the presented environments stay rather simple and are not suitable for highly complex tasks such as control learning.

Figure 6.7 offers a visual snapshot of one such experiment conducted in our laboratory, featuring the Escort environment. The drone highlighted with a yellow circle symbolizes the moving target, while all the other drones represent the agents controlled by the multi-agent policy. For a clearer view of the results, a video is accessible at https://www.youtube.com/watch?v=4FeTjZnpgJI&ab_channel=FlorianFelten.

6.6 Summary

This chapter presented the task of learning swarming behaviors involving trade-offs using reinforcement learning. We presented novel scenarios involving groups of drones striving to learn formation behaviors. In these scenarios, agents exhibited different compromise behaviors depending on two objectives: they need to learn how to gather around an object while simultaneously maintaining a safe distance from one another. To address this challenge, we introduced MOMAAD, a novel multi-objective multi-agent reinforcement learning algorithm designed to learn a Pareto set of multi-agent policies by fully leveraging hardware acceleration.

A key distinction of MOMAAD is its avoidance of manual tuning of weights, which is typically required in current approaches via a trial-and-error process. Our findings demonstrate that MOMAAD can learn numerous trade-offs between these objectives for effective swarm control in a few seconds. In contrast to traditional approaches executed in CPU-based settings, MOMAAD delivered exceptional acceleration, ultimately reaching a performance level 2700 times faster on a standard desktop computer. This swift, Python-based implementation offers a compelling alternative to more intricate methods aimed at enhancing sample efficiency to learn in challenging domains. Looking ahead, we believe that this research paves the way for wider adoption of fully accelerated reinforcement learning where feasible.

6.7 Conclusion

This part of the thesis expanded our work to settings involving multiple agents collaborating to achieve a task. In these new settings, challenges arise due to the need to optimize multiple objectives while simultaneously enabling multiple agents to learn. Our initial contributions focused on laying a solid foundation for future research in the emerging field of MOMARL. To this end, we introduced MOMAland, an open-source platform containing standard APIs and environments tailored for MOMARL research. Additionally, MOMAland incorporates several algorithms capable of learning Pareto sets of multi-agent policies in these complex settings. Notably, these algorithms enable the direct utilization of knowledge from both MORL and MARL, akin to our previous work of transferring concepts from MOO and RL into MORL.

Subsequently, we explored a real-life application of MOMARL to demonstrate the practical deployment of these novel methods. Our study focused on controlling a swarm of drones tasked with surrounding a potentially moving target while maintaining a safe distance from each other. Unlike traditional methods that rely on a trial-and-error process to assign importance to each objective, our approach empowers decision-makers to choose optimal behaviors after the learning process, based on observed outcomes in a simulator. This approach enhances algorithm learning efficiency by providing awareness during the learning process of the need to acquire multiple trade-off policies. Notably, our real-world experiments highlight parallelization and decomposition as promising avenues for future research, offering significant speed improvements.

Published Work

The environments presented in this part have been included in MOMAland:

- Florian Felten, Umut Ucak, Hicham Azmani, Gao Peng, Willem Röpke, Hendrik Baier, Patrick Mannion, Diederik M. Roijers, Jordan K. Terry, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, and Roxana Rădulescu. MOMAland: A Set of Benchmarks for Multi-Objective Multi-Agent Reinforcement Learning. 2024b

The idea of accelerated decomposition and the associated algorithm (MOMAAD) are novel contributions introduced in this thesis.

Part IV

Conclusion

Chapter 7

Conclusion

Contents

In this dissertation, we discussed the extension of reinforcement learning to tackle problems where (a) the agent has to make compromises among multiple conflicting objectives, and (b) multiple agents can interact in a single environment while making compromises among multiple objectives. In contrast to conventional approaches that typically scalarize the objectives into a single value with designated importance assigned by a weight vector, our research focused on acquiring diverse policies that result in a range of trade-offs among objectives. We saw that learning algorithms can be more efficient in such settings as they recognize the need to learn varying trade-offs. This chapter concludes with a final discussion on the presented contributions, and gives insight on future research directions and current limitations of the state of the art.

7.1 Discussion

This part summarizes our work and refers to the original research questions identified in Section 1.

Q1: What are the commonalities and differences between the fields of MOO, RL, and MORL? Our work started by a comprehensive review of the current state-of-the-art methods for solving MORL problems (Section 2.3). Based on this, we identified recurring patterns coming in various MORL algorithms, and identified those patterns in the domains of single-objective RL and multi-objective optimization. By establishing

a taxonomy to categorize these commonalities and differences (Section 2.3.4), we laid the groundwork for integrating strategies from MOO and RL into MORL frameworks, as well as enabled to precisely describe in which domain the contributions lie. Ultimately, as demonstrated in MOMAland (Chapter 5), techniques seen in MORL research extend to MOMARL as well.

Q2: To what extent can existing methods and knowledge from older fields be applied to MO(MA)RL? After identifying the common ideas between MORL, RL and MOO, one direction of our work has been to show that existing methods from older fields such as MOO can be readily applied to MORL. We believe that harnessing the wealth of knowledge accumulated in fields with decades of study promises enduring benefits for the advancement of MORL and MOMARL. Thus, dedicating a significant portion of this work to facilitate the "transfer of ideas" emerged as a key strategy for accelerated progress and innovation in the realms of MO(MA)RL.

First, we showed that metaheuristics strategies can be used as exploration-exploitation strategies in MORL and improve over the current state of the art (Section 3.1). Second, we derived a general framework for MORL based on decomposition (Section 3.2). This framework is grounded on our thorough analysis and presented taxonomy. Being modular, it allows using different concepts from the MOO and RL domains to form new MORL algorithms. To showcase its modularity, we instantiated our framework to solve varied problems with concave or convex Pareto fronts and showed it has similar performance to current methods.

Next, within MOMAland, we demonstrated the applicability of existing MORL and MARL methods to solve MOMARL problems. These approaches involved leveraging decomposition techniques and solving single-objective subproblems using MARL algorithms or centralizing agents and applying MORL algorithms. Despite their simplicity, these methods serve as robust baselines for future MOMARL investigations and allow for easy integration of novel concepts coming from RL, MOO, MORL, or MARL.

Q3: What constitutes a rigorous scientific methodology for conducting MO(MA)RL research? Another point of the presented work was to make MO(MA)RL research more rigorous and reliable. Upon embarking on this research journey, the fields of MORL and MOMARL lacked standardized practices for conducting research and defining valid outcomes. Questions such as defining standard benchmark environments, selecting suitable evaluation metrics, and determining hyperparameter values were unresolved. These had severe impacts on the reproducibility aspects and scientific validity of MO(MA)RL research.

For this, we first introduced a suite of tools aimed at aiding MORL researchers to conduct their work and engineers wanting to apply MORL to real-world scenarios. The suite contains MO-Gymnasium (MORL benchmark environments, Section 4.1), MORL-Baselines (state-of-the-art algorithms implementations, Section 4.2), open training results (Section 4.3), and hyperparameter optimization for MORL (Section 4.4). Currently, with over 100,000 downloads for MO-Gymnasium, these tools are extensively used by the MORL community and significantly influence the empirical validation of MORL research.

Subsequently, we expanded our work to the nascent field of multi-objective multi-agent RL, aiming to establish foundational principles and tools for this field. For this, we introduced MOMAland, comprising a collection of benchmark environments for MOMARL under standardized APIs as well as robust baseline algorithms (Section 5.1). We believe the introduction of reliable benchmarks will be a pillar to the advancement of this field of research, similar to the impact MO-Gymnasium and MORL-Baselines have on MORL.

Q4: What challenges hinder the deployment of MO(MA)RL methods in real-world scenarios? To address this, we wish to reiterate the impact of our open-source tools in streamlining the deployment of such techniques. As an example, non-specialists in MORL currently can utilize the MO-Gymnasium API to instantiate their environments and then rely on MORL-Baselines to derive solutions.

Moreover, we endeavored to apply our accumulated knowledge and expertise to practical scenarios. For this purpose, we explored the utilization of MOMARL in learning swarming behaviors for coordinating multiple drones tasked with surrounding a potentially moving target (Chapter 6). Notably, we adapted the concept of decomposition to parallelize learning processes on hardware-accelerated devices, resulting in significant acceleration of the learning process by factors exceeding 2000. Subsequently, the acquired policies were deployed on real drones to validate the efficacy of our approach. This project showed the entire process of deployment of MOMARL methods with unknown user preferences in the real world.

7.2 Future research

Given the recent emergence of multi-objective RL and its applicability to many scenarios, we anticipate numerous future research within this domain. Furthermore, the nascent field of multi-objective multi-agent RL remains relatively underexplored. This section lists some of the challenges we have identified within these domains.

7.2.1 Enhancing Algorithms

Because of their demonstrated centrality within the thesis, it is evident that improving MORL algorithms may lead to benefits in various domains. We identify four research directions in this line.

7.2.1.1 Deep Pareto-based MORL

As mentioned in Section 2.3.3, Pareto-based MORL methods are currently limited to tabular settings. Yet, as shown in Section 4.3, these methods show strong performance where applicable. We believe that the adaptation of these methods to deep MORL settings might lead to interesting results in more challenging domains. For this, we believe that recent advances in sequential neural networks such as recurrent neural networks [Hochreiter and Schmidhuber, 1997] or Transformers [Vaswani et al., 2017] may help to learn Pareto fronts.

7.2.1.2 Reducing the Objective Space

While a significant amount of MORL research targets learning a PF spanning the entire objective space, there are applications where users are only interested in a small subregion of this space. One interesting way to solve this is to learn the user's preferences through elicitation while learning the PS of policies [Zintgraf et al., 2018]. Another way is to put constraints on the objective space or target specific regions [Vamplew et al., 2017b].

7.2.1.3 More MORL/D Variants

In Section 3.2, we presented a generic framework allowing to construct new MORL algorithms by making atomic changes in building blocks exposed in the introduced taxonomy. This framework has been instantiated in different ways to show how versatile it is. We believe this work could be extended by studying additional

techniques, such as new cooperation techniques like crossover of policies (also called soups of models in the recent literature [Wortsman et al., 2022]). Naturally, other scalarization functions and ways to generate weight vectors or reference points (*e.g.*, Röpke et al. [2024]) may also lead to improvements of the current state of MORL. Finally, we believe that combining the MORL/D framework and an hyperparameter optimizer may give promising results in automatically generating MORL algorithms tailored to given problems, forming an "AutoMORL" solver.

7.2.1.4 Accelerated MO(MA)RL

In Section 6.3, we presented the concept of accelerated decomposition, where subproblems are solved in parallel on different cores from an accelerated device. When presenting MOMAAD as an embarrassingly parallel algorithm, we opposed it to cooperative algorithms such as MORL/D presented earlier (Figure 6.2). We believe that these paradigms are not exclusively separate and there exists a middle ground between these techniques. For instance, relying on cooperation techniques that do not necessitate thread synchronization, such as relying on conflict-free replicated data types (CRDTs, [Shapiro et al., 2011]). Moreover, the proposed accelerated algorithm relies on equally spaced weights in the objective space, this approach might not be the most efficient, as the scale between objectives are often different in practice, making numerous weight vectors lead to similar regions of the objective space. As explained earlier (Section 2.2.5), there exist advanced weight vector generation techniques, such as OLS [Roijers et al., 2015b], which allow for better allocation of the computational budget. However, these techniques often rely on the current estimation of the PF, requiring synchronization points, and producing only a few weight vector candidates. We believe that these techniques may be extended to produce a larger number of weight vectors and allow for batch learning of policies in a parallel manner.

7.2.2 Better Experimental Settings

In Chapters 4 and 5, we presented our contributions to improve the current state of the experimental process for MO(MA)RL. We believe that there is still work ahead in this field too.

7.2.2.1 Better Metrics

We identified in various places (*e.g.*, Sections 3.2 and 4.3) that multi-objective metrics can be confusing in some cases, such as when the scale of objectives are different. While the metrics provided in MORL-Baselines allow mitigating these problems by relying on different metrics quantifying diversity and convergence, we believe there are other existing metrics that would be interesting to include, as studied in Audet et al. [2021]. Moreover, one of the current limitations of the MORL-Baselines metrics is that they are computed in an online manner. Hence, it is impossible to normalize the learned PFs or compute convergence metrics if the optimal PF is unknown. We believe a simple yet powerful addition to the library would be a script to compute metrics in an offline manner, *i.e.*, after the learning process, then one could combine all learned PFs from all runs and form an optimal PF to compute additional metrics and normalize PFs.

7.2.2.2 An Empirical Study of What Matters in MORL

While our contributions represent significant progress in the application and deployment of MORL on a larger scale, we believe there are additional studies to be conducted. Indeed, the results showcased in Section 4.3 were presented to illustrate the capabilities of the introduced tools and cannot be used to draw statistically

significant conclusions on the performance of algorithms. Additionally, these runs were performed with default hyperparameters, which may result in suboptimal learning performance, as demonstrated in Section 4.4. However, the suite of tools presented in Chapter 4 enables a rigorous large-scale empirical study akin to Andrychowicz et al. [2021]. This would facilitate fair comparisons between MORL algorithms, an in-depth study of the influence of hyperparameters using sensitivity analysis, and the establishment of a central point for comparing solving methods to understand what truly works. The biggest obstacle to performing such a study is the computational needs it entails.

7.2.3 Enhancing HPO for MORL

In Section 4.4, we presented an initial algorithm allowing to perform hyperparameter optimization for MORL. While early results are promising, our implementation could be further improved in various ways. For example, one could improve the optimizer by including more advanced techniques such as hyperband [Li et al., 2018] or by distributing the search phase over various computing nodes. Moreover, the set of tools for sensitivity analysis of the optimization process to gain better understanding of the hyperparameters' influence could be enhanced by relying on the extensive literature from the supervised learning community. Finally, it is important to evaluate how many search seeds are sufficient to determine high-performing hyperparameters as a general practice.

7.2.4 Expanding MOMARL

The second part of the thesis presented contributions in the nascent field of MOMARL. While conducting our research in this field, we identified a few key research topics for the future.

7.2.4.1 Solution Concepts for MOMARL Settings

The last part of the thesis studied the extension of MORL to multi-agent settings where agents are assumed to be homogeneous and cooperative. These assumptions allow falling back to the same solution concepts as single-agent MORL. However, there are other settings which have not been studied yet. As briefly discussed in Section 5.1, a central open question in MOMARL is the definition of solution concepts when agents are in adversarial or mixed settings and their utility is unknown. Moreover, we currently have little research on settings where each agent has its own preferences [Rădulescu et al., 2020].

7.2.4.2 Novel MOMARL Algorithms

As we have mentioned before, only a few studies have concentrated on learning in environments with multiple objectives and multiple agents. In Section 5.1, we introduced solving methods that rely on centralization and decomposition, but these approaches have their drawbacks. Centralization struggles to scale with the increasing number of agents, while decomposition does not effectively scale with the growing number of objectives. It is vital to tackle these two problems to make significant advancements in the field of MOMARL algorithms. Moreover, we believe that the recent rise of interest in MORL might also expand to MOMARL settings due to their necessity in some applications. Therefore, we believe novel MOMARL algorithms will emerge in the upcoming years. These will probably borrow concepts from the MORL and MARL worlds, as we have shown in Section 5.5.

7.2.5 More Applications

In recent years, MORL has garnered increasing attention from the research community. With MOMARL now well-defined and built upon robust foundations, it stands poised to attract greater interest. However, despite this progress, there have been relatively few real-world applications of these techniques thus far. We believe that our software tools will ease the applicability of such techniques and anticipate that the application of these methods will not only yield valuable insights but also raise important open questions for the research community to address.

7.3 Contributions

This section summarizes our contributions. First, peer-reviewed articles are enumerated, then open-source software and outreach initiatives are listed.

7.3.1 Peer-Reviewed Contributions

Below are all the reviewed contributions done during the course of the PhD degree, in historical order:

- Conference paper – nominated for "best student paper award": **Florian Felten**, Grégoire Danoy, El-Ghazali Talbi, and Pascal Bouvry. Metaheuristics-based Exploration Strategies for Multi-Objective Reinforcement Learning. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence* (ICAART), pages 662–673. SCITEPRESS - Science and Technology Publications, 2022. ISBN 978-989-758-547-0. doi: 10.5220/0010989100003116;
- Extended abstract: **Florian Felten**, El-Ghazali Talbi, and Grégoire Danoy. MORL/D: Multi-Objective Reinforcement Learning based on Decomposition. In *Proceedings of the International Conference in Optimization and Learning* (OLA), 2022;
- Demo paper: Lucas N. Alegre, **Florian Felten**, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, Ana L. C. Bazzan, and Bruno C. da Silva. MO-Gym: A Library of Multi-Objective Reinforcement Learning Environments. In *Proceedings of the 34th Benelux Conference on Artificial Intelligence* (BNAIC/BeNeLearn), 2022;
- Conference paper: **Florian Felten***, Lucas N. Alegre*, Ann Nowé, Ana L. C. Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno C. da Silva. A Toolkit for Reliable Benchmarking and Research in Multi-Objective Reinforcement Learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems* (NeurIPS), 2023;
- Workshop paper: **Florian Felten**, Daniel Gareev, El-Ghazali Talbi, and Grégoire Danoy. Hyperparameter Optimization for Multi-Objective Reinforcement Learning, in *Multi-Objective Decision Making Workshop* (MODeM), 2023;
- Journal paper: **Florian Felten**, El-Ghazali Talbi, and Grégoire Danoy. Multi-Objective Reinforcement Learning Based on Decomposition: A Taxonomy and Framework. *Journal of Artificial Intelligence Research* (JAIR), 79:679–723, 2024. ISSN 1076-9757. doi: 10.1613/jair.1.15702. URL https://www.jair.org/index.php/jair/article/view/15702;

- Conference paper: Shengyi Huang*, Quentin Gallouédec*, **Florian Felten**, Antonin Raffin, Rousslan Fernand Julien Dossa, Yanxiao Zhao, Ryan Sullivan, Viktor Makoviychuk, Denys Makoviichuk, Mohamad H. Danesh, Cyril Roumégous, Jiayi Weng, Chufan Chen, Md Masudur Rahman, João G. M. Araújo, Guorui Quan, Daniel Tan, Timo Klein, Rujikorn Charakorn, Mark Towers, Yann Berthelot, Kinal Mehta, Dipam Chakraborty, Arjun KG, Valentin Charraut, Chang Ye, Zichen Liu, Lucas N. Alegre, Alexander Nikulin, Xiao Hu, Tianlin Liu, Jongwook Choi, and Brent Yi. Open RL Benchmark: Comprehensive Tracked Experiments for Reinforcement Learning, 2024. URL http://arxiv.org/abs/2402.03046. arXiv:2402.03046 [cs] (in review);
- Journal paper: **Florian Felten**, Umut Ucak, Hicham Azmani, Gao Peng, Willem Röpke, Hendrik Baier, Patrick Mannion, Diederik M. Roijers, Jordan K. Terry, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, and Roxana Rădulescu. MOMAland: A Set of Benchmarks for Multi-Objective Multi-Agent Reinforcement Learning. 2024 (in review).

7.3.2 Open-Source Software Contributions

- MO-Gymnasium: https://mo-gymnasium.farama.org/;
- MORL-Baselines: https://lucasalegre.github.io/morl-baselines/;
- Open RL Benchmark: https://github.com/openrlbenchmark/openrlbenchmark;
- MOMAland: https://momaland.farama.org/;
- CrazyRL: https://github.com/ffelten/CrazyRL.

7.3.3 Outreach

- Talk @ University of Luxembourg. Multi-Objective Reinforcement Learning. 2022;
- Demo @ FNR Researchers' days, Luxembourg. Controlling robots with AI. 2022;
- Demo @ Partnership days SnT, Luxembourg. Autonomous Swarms of Drones. 2023;
- Talk @ International Conference in Optimization and Learning (OLA). CrazyRL : A Multi-Agent Reinforcement Learning library for flying Crazyflie drones. 2023;
- Talk @ Vrije Universiteit Brussels. Automated Generation of Heuristics for Swarm Control. 2023;
- FNR Spotlight on Young Researchers: The challenge of getting autonomous systems to work together seamlessly, video. 2023;
- Demo @ Innovation days European Investment Bank, Luxembourg. Autonomous Swarms of Drones. 2023;
- Talk @ University of Luxembourg. MORL in Practice: From Craftsmanship to Science. 2024.

Appendix A

Existing Works Classified in the MORL/D Taxonomy

Table A.1 presents existing MORL work classified according to the proposed MORL/D taxonomy (Figure 2.12). This demonstrates the modularity of the proposed framework and offers a high-level view of the current state of the art in MORL. The table is separated by a double vertical line showing again the boundaries between traits inherited from MOO and RL. For each trait, a column presents the instantiation choice made in each paper.

For instance, in the first line, the work of [Roijers and Whiteson, 2017] dynamically assigns weight vectors based on Optimistic Linear Support (OLS). As a cooperation mechanism, the algorithm reuses (transfers) the knowledge from the closest already trained policy to hot-start the training of a new policy. The algorithm relies on n tabular representations, where n is the number of desired policies. The Bellman update used is based on a Partially Observable MDP solver (POMDP). Finally, the sampling strategy proposes to follow the policy currently being trained (with its internal exploration technique).

All the works referenced in the table make use of the weighted sum scalarization. Thus, scalarization and reference point columns have been omitted from the table since they would bring little information. In general, non-linear scalarization schemes are understudied when compared to the weighted sum. Additionally, for space reasons, population selection and archive have not been represented either. In both traits, the work of [Xu et al., 2020a] is particularly interesting.

Reference	MOO					RL				
	Weight vectors		Cooperation					Buffer		
	When?	How?	Neighb.	Mechanism	Trigger	Regression structure	Policy improv.	Neighb.	Storage & Sampling Strategy	Sampling strategy
[Roijers et al., 2015a]	Dynamic	Adaptive - OLS	Single - Closest weight	Transfer	Periodic	$n\times$ Tabular	Scalarized POMDP solver	/	/	Policy following
[Mossalam et al., 2016]	Dynamic	Adaptive - OLS	Single - Closest weight	Transfer	Periodic	$n\times$ DNN + MO reg.	Scalarized DQN	Indep.	Recency + Uniform	Policy following
[Chen et al., 2020]	Static	Manual	All	Shared buffer Shared layers	Continuous	$n\times$ DNN	Scalarized SAC	All	Recency + Uniform	Parallel policy following
[Yang et al., 2019]	Dynamic	Random	All	CR	Continuous	1$\times$ DNN	Envelope DQN	All	HER + Recency + Uniform	Policy following
[Xu et al., 2020a]	Dynamic	Uniform	None	None	None	$n\times$ DNN + MO reg.	Scalarized PPO	Indep.	Recency + Uniform	Policy following
[Abels et al., 2019]	Dynamic	Random	All	CR	Continuous	1$\times$ DNN + MO reg.	Scalarized, Multi-weights DQN	All	HER + PER (Diversity)	Policy following
[Alegre et al., 2023]	Dynamic	Adaptive - GPI-LS	All	CR Shared model	Continuous	1$\times$ DNN + MO reg.	Scalarized, Multi-weights DQN or TD3	All	HER + PER (GPI)	Policy following
[Castelletti et al., 2013]	Dynamic	Random	All	CR	Continuous	1$\times$ Trees	Scalarized FQI	/	/	Historical dataset

TABLE A.1: A non-exhaustive list of MORL works classified according to the MORL/D taxonomy. OLS = Optimistic Linear Support, DNN = Deep Neural Network, POMDP = Partially Observable MDP, MO reg. = Multi-objective regression, CR = Conditioned Regression, HER = Hindsight Experience Replay, PER = prioritized experience replay, GPI = Generalized Policy Improvement.

Appendix B

Weights and Biases Dashboards

This appendix provides screenshots of the Weights and Biases Dashboards visualizing the results of training MORL-Baselines algorithms on MO-Gymnasium environments. They are hosted in Open RL Benchmark [Huang et al., 2024].

Figure B.1 is a screenshot of the overview dashboard for a given run. It provides information such as the name of the run, git commit used to produce the results, duration of the run, hardware and operating system used, Python version, and command line used to launch the job.

Figure B.1 illustrates the hyperparameter values recorded for a given run. This, along with the git commit and command line, allows for reproducibility of our results.

Figure B.3 gives an overview of the training data hosted in ORLB. Importantly, it is possible to group and filter the runs for a given environment and by algorithm. The dashboards expose the multi-objective performance indicators discussed in Section 2.2.3 and PF approximation of different algorithms.

FIGURE B.1: Overview of a run in our dashboards.

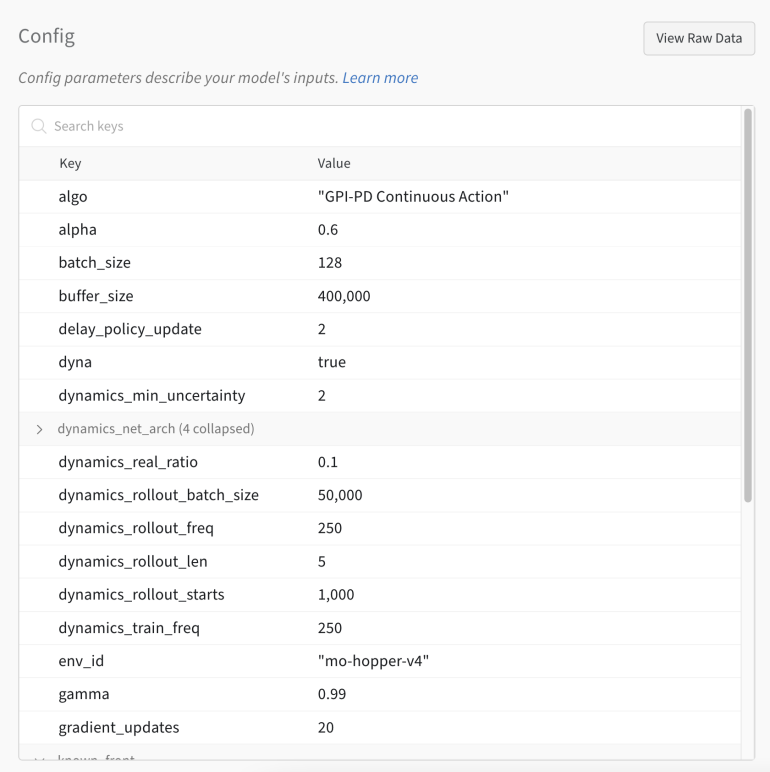


FIGURE B.2: Hyperparameter values for a given run in our dashboards.

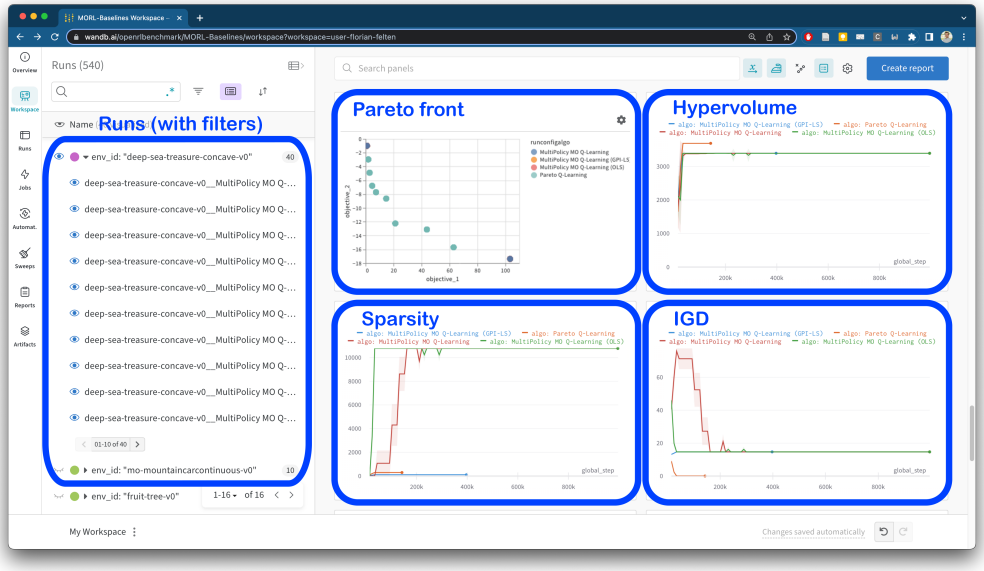


FIGURE B.3: Overview of performance indicator values over the training phase in our dashboards.

Appendix C

CrazyRL appendices

C.1 Reproducibility

For the results involving multi-objective problems, we used the following environment with MOMAAD to generate the Pareto front (Figure 6.5):

```
num_drones = 8
env = Surround(
    num_drones=num_drones,
    init_flying_pos=jnp.array(
        [
            [0.0, 0.0, 1.0],
            [0.0, 1.0, 1.0],
            [1.0, 0.0, 1.0],
            [1.0, 2.0, 2.0],
            [2.0, 0.5, 1.0],
            [2.0, 2.5, 2.0],
            [2.0, 1.0, 2.5],
            [0.5, 0.5, 0.5],
        ]
    ),
    target_location=jnp.array([1.0, 1.0, 2.0]),
    multi_obj=True,
    size=5,
)
```

To generate the plots comparing the time to train one or more policies (Figure 6.3 and Table 6.2), we used the following:

```
num_drones = 2
# Figure and table
env = Surround(
```

```
    num_drones=num_drones,
    init_flying_pos=jnp.array(
        [
            [0.0, 0.0, 1.0],
            [0.0, 1.0, 1.0],
        ]
    ),
    target_location=jnp.array([1.0, 1.0, 2.0]),
    multi_obj=True,
    size=5,
)

# Figure only
env = Escort(
    num_drones=num_drones,
    init_flying_pos=jnp.array(
        [
            [0.0, 0.0, 1.0],
            [0.0, 1.0, 1.0],
        ]
    ),
    init_target_location=jnp.array([1.0, 1.0, 2.0]),
    multi_obj=True,
    size=5,
    final_target_location=jnp.array([-2.0, -2.0, 1.0]),
)

env = Catch(
    num_drones=num_drones,
    init_flying_pos=jnp.array(
        [
            [0.0, 0.0, 1.0],
            [0.0, 1.0, 1.0],
        ]
    ),
    init_target_location=jnp.array([1.0, 1.0, 2.0]),
    multi_obj=True,
    size=5,
    target_speed=0.15,
)
```

C.1.1 Real-conditions

This section gives the parameters used to generate and execute the policies shown in the video:

Surround. For surround with 5 agents:

```
env = Surround(
    drone_ids=drones_ids,
    render_mode="real",
    init_flying_pos=jnp.array(
        [
            [-1.0, 0.0, 1.0],
            [-1.0, 0.5, 1.5],
            [0.0, 1.0, 1.0],
            [0.5, 0.0, 0.5],
            [0.5, -0.5, 1.5],
        ]
    ),
    target_location=jnp.array([0.0, 0.5, 1.5]),
    target_id=target_id,
    swarm=swarm,
)
```

Escort. For escort with 4 agents:

```
env = Escort(
    drone_ids=drones_ids,
    render_mode="real",
    init_flying_pos=jnp.array(
        [
            [-0.8, 0.5, 0.5],
            [1.0, 0.5, 1.5],
            [0.5, 0.0, 0.5],
            [0.5, -0.5, 1.0],
        ]
    ),
    init_target_location=jnp.array([-0.1, 0.6, 1.1]),
    final_target_location=jnp.array([1.2, -1.3, 2.3]),
    target_id=target_id,
    swarm=swarm,
)
```

Catch (4). For catch with 4 agents:

```
env = Catch(
    drone_ids=drones_ids,
    render_mode="real",
    init_flying_pos=jnp.array(
        [
```

```
            [-0.8, 0.5, 0.5],
            [1.0, 0.5, 1.5],
            [0.5, 0.0, 0.5],
            [0.5, -0.5, 1.0],
        ]
    ),
    init_target_location=jnp.array([0.0, 0.5, 1.5]),
    target_speed=0.1,
    size=1.3,
    target_id=target_id,
    swarm=swarm,
)
```

Catch (8). For catch with 8 agents (bonus):

```
env = Catch(
    drone_ids=np.arange(8),
    render_mode="human",
    init_flying_pos=jnp.array(
        [
            [-0.7, -0.5, 1.5],
            [-0.8, 0.5, 0.5],
            [1.0, 0.5, 1.5],
            [0.5, 0.0, 0.5],
            [0.5, -0.5, 1.0],
            [2.0, 2.5, 2.0],
            [2.0, 1.0, 2.5],
            [0.5, 0.5, 0.5],
        ]
    ),
    init_target_location=jnp.array([0.0, 0.5, 1.5]),
    target_speed=0.15,
    size=5,
)
```

C.2 Differences between MAPPO and MOMAAD

Here, we showcase the essential code differences to transform a full JAX MARL implementation into a fully accelerated MOMARL algorithm. This demonstrates that it is not necessarily complex to benefit from acceleration using such abstraction. The first step is to extend the train function to be parameterized by weights. The second step is to use our provided wrapper, which transforms the rewards into scalar automatically, given some weights. The last step involves the usage of `vmap` over the train function to add a batch dimension. This batch dimension allows to perform parallel training on multiple weights.

```
@@ -135 +135 @@
```

```
-   def train(key: chex.PRNGKey):
+   def train(key: chex.PRNGKey, weights: jnp.ndarray):

@@ -161 +161 @@
    env = AddIDToObs(env)
+   env = LinearizeReward(env, weights)
    env = AutoReset(env)

@@ -483, +465 @@
    start_time = time.time()
-   train_jit = jax.jit(make_train(args))
-   out = jax.block_until_ready(train_jit(rng))
+   weights = jnp.array(
+       equally_spaced_weights(2, NUM_WEIGHTS)
+   )
+   train_vjit = jax.jit(
+           jax.vmap(make_train(args), in_axes=(None, 0)),
+   )
+   out = jax.block_until_ready(train_vjit(rng, weights))
    print(f"total time: {time.time() - start_time}")
```

Bibliography

Axel Abels, Diederik Roijers, Tom Lenaerts, Ann Nowé, and Denis Steckelmacher. Dynamic Weights in Multi-Objective Deep Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 11–20. PMLR, May 2019. URL https://proceedings.mlr.press/v97/abels19a.html. ISSN: 2640-3498.
10 citations in pages 29, 30, 33, 56, 62, 65, 69, 79, 93, and 120.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=uqv8-U4lKBe.
3 citations in pages 3, 61, and 62.

Ishfaq Ahmad, Sanjay Ranka, and Samee Ullah Khan. Using game theory for scheduling tasks on multi-core processors for simultaneous optimization of performance and energy. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–6, 2008. doi: 10.1109/IPDPS.2008.4536420.
One citation in page 39.

Ines Alaya, Christine Solnon, and Khaled Ghedira. Ant Colony Optimization for Multi-objective Optimization Problems. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 450–457, Patras, Greece, October 2007. IEEE Computer Society. URL https://hal.archives-ouvertes.fr/hal-01502167.
One citation in page 16.

Lucas N. Alegre, Florian Felten, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, Ana LC Bazzan, and Bruno C. da Silva. MO-Gym: A Library of Multi-Objective Reinforcement Learning Environments. In *Proceedings of the 34th Benelux Conference on Artificial Intelligence BNAIC/Benelearn*, 2022.
5 citations in pages 34, 46, 52, 63, and 86.

Lucas N Alegre, Ana L C Bazzan, Diederik M Roijers, and Ann Nowé. Sample-Efficient Multi-Objective Learning via Generalized Policy Improvement Prioritization. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems*, 2023.
10 citations in pages 29, 30, 34, 67, 68, 69, 81, 93, 95, and 120.

Abdullah M. Almasoud and Ahmed E. Kamal. Multi-Objective Optimization for Many-to-Many Communication in Cognitive Radio Networks. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2015. doi: 10.1109/GLOCOM.2015.7417485.
One citation in page 24.

Fulya Altiparmak, Mitsuo Gen, Lin Lin, and Turan Paksoy. A genetic algorithm approach for multi-objective optimization of supply chain networks. *Computers & Industrial Engineering*, 51(1):196–215, September 2006. ISSN 0360-8352. doi: 10.1016/j.cie.2006.07.011. URL https://www.sciencedirect.com/science/article/pii/S0360835206000763.
One citation in page 24.

Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, April 1967. Association for Computing Machinery. ISBN 978-1-4503-7895-6. doi: 10.1145/1465482.1465560. URL https://dl.acm.org/doi/10.1145/1465482.1465560.
One citation in page 102.

Susan Amin, Maziar Gomrokchi, Harsh Satija, Herke van Hoof, and Doina Precup. A Survey of Exploration Methods in Reinforcement Learning. *arXiv:2109.00157 [cs]*, September 2021. URL http://arxiv.org/abs/2109.00157. arXiv: 2109.00157.
2 citations in pages 13 and 42.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5055–5065, Red Hook, NY, USA, December 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.
One citation in page 33.

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=nIAxjsniDzg.
3 citations in pages 62, 74, and 116.

Charles Audet, Jean Bigeon, Dominique Cartier, Sébastien Le Digabel, and Ludovic Salomon. Performance indicators in multiobjective optimization. *European Journal of Operational Research*, 292(2):397–422, 2021. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.11.016. URL https://www.sciencedirect.com/science/article/pii/S0377221720309620.
One citation in page 115.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martin Arjovsky, Alexander Pritzel, Andrew Bolt, and Charles Blundell. Never Give Up: Learning Directed Exploration Strategies. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Sye57xStvB.
One citation in page 44.

Oliver Bandte and Sergey Malinchik. A Broad and Narrow Approach to Interactive Evolutionary Design – An Aircraft Design Example. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation – GECCO 2004*, Lecture Notes in Computer Science, pages 883–895, Berlin, Heidelberg, 2004. Springer. ISBN 978-3-540-24855-2. doi: 10.1007/978-3-540-24855-2_102.
One citation in page 24.

Eugenio Bargiacchi, Timothy Verstraeten, Diederik Roijers, Ann Nowé, and Hado van Hasselt. Learning to Coordinate with Coordination Graphs in Repeated Single-Stage Multi-Agent Decision Problems. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 482–490. PMLR, July 2018. URL https://proceedings.mlr.press/v80/bargiacchi18a.html.
One citation in page 88.

Andre Barreto, Will Dabney, Remi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor Features for Transfer in Reinforcement Learning. In I. Guyon, U. Von Luxburg, S. Bengio,

H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/350db081a661525235354dd3e19b8c05-Paper.pdf.
One citation in page 68.

Levent Bayındır. A review of swarm robotics tasks. *Neurocomputing*, 172:292–321, January 2016. ISSN 0925-2312. doi: 10.1016/j.neucom.2015.05.116. URL https://www.sciencedirect.com/science/article/pii/S0925231215010486.
3 citations in pages 2, 97, and 98.

Lukas Biewald. Experiment Tracking with Weights and Biases, 2020. URL https://www.wandb.com/.
2 citations in pages 62 and 79.

Martin Binder, Julia Moosbauer, Janek Thomas, and Bernd Bischl. Multi-objective hyperparameter tuning and feature selection using filter ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, pages 471–479, New York, NY, USA, June 2020. Association for Computing Machinery. ISBN 978-1-4503-7128-5. doi: 10.1145/3377930.3389815. URL https://dl.acm.org/doi/10.1145/3377930.3389815.
One citation in page 76.

Mauro Birattari, Antoine Ligot, Darko Bozhinoski, Manuele Brambilla, Gianpiero Francesca, Lorenzo Garattoni, David Garzón Ramos, Ken Hasselmann, Miquel Kegeleirs, Jonas Kuckling, Federico Pagnozzi, Andrea Roli, Muhammad Salman, and Thomas Stützle. Automatic Off-Line Design of Robot Swarms: A Manifesto. *Frontiers in Robotics and AI*, 6, July 2019. doi: 10.3389/frobt.2019.00059.
One citation in page 97.

Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *WIREs Data Mining and Knowledge Discovery*, 13(2):e1484, 2023. ISSN 1942-4795. doi: 10.1002/widm.1484. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1484. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1484.
2 citations in pages 75 and 76.

J. Blank and K. Deb. pymoo: Multi-Objective Optimization in Python. *IEEE Access*, 8:89497–89509, 2020.
3 citations in pages 52, 69, and 105.

Julian Blank, Kalyanmoy Deb, Yashesh Dhebar, Sunith Bandaru, and Haitham Seada. Generating Well-Spaced Points on a Unit Simplex for Evolutionary Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation*, 25(1):48–60, February 2021. ISSN 1941-0026. doi: 10.1109/TEVC.2020.2992387. Conference Name: IEEE Transactions on Evolutionary Computation.
6 citations in pages 22, 54, 69, 93, 94, and 105.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.
2 citations in pages 98 and 101.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, June 2016. URL http://arxiv.org/abs/1606.01540. arXiv:1606.01540 [cs].
2 citations in pages 62 and 63.

Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12): 1695–1724, December 2013. ISSN 1476-9360. doi: 10.1057/jors.2013.71. URL https://doi.org/10.1057/jors.2013.71.
One citation in page 24.

Thomas Cassimon, Reinout Eyckerman, Siegfried Mercelis, Steven Latré, and Peter Hellinckx. A Review of the Deep Sea Treasure problem as a Multi-Objective Reinforcement Learning Benchmark. *arXiv:2110.06742 [cs]*, October 2021. URL http://arxiv.org/abs/2110.06742. arXiv: 2110.06742.
One citation in page 62.

A. Castelletti, F. Pianosi, and M. Restelli. A multiobjective reinforcement learning approach to water resources systems operation: Pareto frontier approximation in a single run. *Water Resources Research*, 49(6):3476–3486, 2013. ISSN 1944-7973. doi: 10.1002/wrcr.20295. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/wrcr.20295. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wrcr.20295.
2 citations in pages 33 and 120.

Diqi Chen, Yizhou Wang, and Wen Gao. Combining a gradient-based method and an evolution strategy for multi-objective reinforcement learning. *Applied Intelligence*, 50, October 2020. doi: 10.1007/s10489-020-01702-7.
5 citations in pages 29, 30, 31, 33, and 120.

Xiao-long Chen, Jun-qing Li, and Ying Xu. Q-learning based multi-objective immune algorithm for fuzzy flexible job shop scheduling problem considering dynamic disruptions. *Swarm and Evolutionary Computation*, 83:101414, December 2023. ISSN 2210-6502. doi: 10.1016/j.swevo.2023.101414. URL https://www.sciencedirect.com/science/article/pii/S2210650223001876.
One citation in page 34.

Carlos A. Coello Coello and Margarita Reyes Sierra. A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm. In Raúl Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Humberto Sossa, editors, *MICAI 2004: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 688–697, Berlin, Heidelberg, 2004. Springer. ISBN 978-3-540-24694-7. doi: 10.1007/978-3-540-24694-7_71.
One citation in page 17.

Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning, 2016. URL http://pybullet.org.
2 citations in pages 99 and 108.

Piotr Czyżżak and Adrezej Jaszkiewicz. Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, 7(1):34–47, 1998.
3 citations in pages 16, 22, and 23.

Indraneel Das and J. Dennis. Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. *SIAM Journal on Optimization*, 8, July 2000. doi: 10.1137/S1052623496307510.
2 citations in pages 22 and 93.

Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuska. Improved deep reinforcement learning for robotics through distribution-based experience retention. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3947–3952, Daejeon, South Korea, October 2016. IEEE. ISBN 978-1-5090-3762-9. doi: 10.1109/IROS.2016.7759581. URL http://ieeexplore.ieee.org/document/7759581/.
One citation in page 13.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002. ISSN 1941-0026. doi: 10.1109/4235.996017. Conference Name: IEEE Transactions on Evolutionary Computation.
2 citations in pages 19 and 30.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, February 2022. ISSN 1476-4687. doi: 10.1038/s41586-021-04301-9. URL https://www.nature.com/articles/s41586-021-04301-9. Number: 7897 Publisher: Nature Publishing Group.
2 citations in pages 1 and 13.

Jérémie Dubois-Lacoste, Manuel López-Ibáñez, and Thomas Stützle. Improving the anytime behavior of two-phase local search. *Annals of Mathematics and Artificial Intelligence*, 61(2):125–154, February 2011. ISSN 1573-7470. doi: 10.1007/s10472-011-9235-0. URL https://doi.org/10.1007/s10472-011-9235-0.
One citation in page 22.

Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in Reinforcement Learning and How To Tune Them. In *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*, June 2023. URL https://openreview.net/forum?id=0Vm8Ghcxmp.
4 citations in pages 51, 75, 76, and 77.

Michael T. M. Emmerich and André H. Deutz. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural Computing*, 17(3):585–609, September 2018. ISSN 1572-9796. doi: 10.1007/s11047-018-9685-y. URL https://doi.org/10.1007/s11047-018-9685-y.
One citation in page 21.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation Matters in Deep RL: A Case Study on PPO and TRPO. In *Proceedings of the Eighth International Conference on Learning Representations*, April 2020. URL https://openreview.net/forum?id=r1etN1rtPB.
2 citations in pages 62 and 66.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1437–1446. PMLR, July 2018. URL https://proceedings.mlr.press/v80/falkner18a.html. ISSN: 2640-3498.
2 citations in pages 76 and 77.

Florian Felten, Grégoire Danoy, El-Ghazali Talbi, and Pascal Bouvry. Metaheuristics-based Exploration Strategies for Multi-Objective Reinforcement Learning:. In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence*, pages 662–673. SCITEPRESS - Science and Technology Publications, 2022. ISBN 978-989-758-547-0. doi: 10.5220/0010989100003116.
2 citations in pages 27 and 31.

Florian Felten, Lucas Nunes Alegre, Ann Nowe, Ana L. C. Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno Castro da Silva. A Toolkit for Reliable Benchmarking and Research in Multi-Objective Reinforcement Learning. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, 2023a.
7 citations in pages 34, 52, 53, 71, 81, 86, and 105.

Florian Felten, Daniel Gareev, El-Ghazali Talbi, and Grégoire Danoy. Hyperparameter Optimization for Multi-Objective Reinforcement Learning, October 2023b. URL http://arxiv.org/abs/2310.16487. arXiv:2310.16487 [cs].
One citation in page 51.

Florian Felten, El-Ghazali Talbi, and Grégoire Danoy. Multi-Objective Reinforcement Learning Based on Decomposition: A Taxonomy and Framework. *Journal of Artificial Intelligence Research*, 79:679–723, February 2024a. ISSN 1076-9757. doi: 10.1613/jair.1.15702. URL https://www.jair.org/index.php/jair/article/view/15702.
One citation in page 67.

Florian Felten, Umut Ucak, Hicham Azmani, Gao Peng, Willem Röpke, Hendrik Baier, Patrick Mannion, Diederik M. Roijers, Jordan K. Terry, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, and Roxana Rădulescu. MOMAland: A Set of Benchmarks for Multi-Objective Multi-Agent Reinforcement Learning. 2024b.
2 citations in pages 88 and 95.

M.J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966. doi: 10.1109/PROC.1966.5273.
One citation in page 101.

J Foerster. *Deep multi-agent reinforcement learning*. PhD Thesis, University of Oxford, 2018.
One citation in page 2.

Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 1146–1155. PMLR, 2017. URL http://proceedings.mlr.press/v70/foerster17b.html.
One citation in page 36.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual Multi-Agent Policy Gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018. ISSN 2374-3468. doi: 10.1609/aaai.v32i1.11794. URL https://ojs.aaai.org/index.php/AAAI/article/view/11794. Number: 1.
One citation in page 36.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation, 2021. URL http://github.com/google/brax.
One citation in page 101.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1587–1596. PMLR, July 2018. URL https://proceedings.mlr.press/v80/fujimoto18a.html. ISSN: 2640-3498.
One citation in page 71.

Luca M. Gambardella and Marco Dorigo. Ant-Q: A Reinforcement Learning approach to the traveling salesman problem. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 252–260. Morgan Kaufmann, San Francisco (CA), January 1995. ISBN 978-1-55860-377-6. doi: 10.1016/B978-1-55860-377-6.50039-6. URL https://www.sciencedirect.com/science/article/pii/B9781558603776500396.
One citation in page 46.

Peter Geibel. Reinforcement Learning for MDPs with Constraints. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006*, Lecture Notes in Computer Science, pages 646–653, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-46056-5. doi: 10.1007/11871842_63.
One citation in page 32.

Wojciech Giernacki, Mateusz Skwierczyński, Wojciech Witwicki, Paweł Wroński, and Piotr Kozierski. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 37–42, August 2017. doi: 10.1109/MMAR.2017.8046794.
2 citations in pages 97 and 108.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
One citation in page 12.

Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In Gita Sukthankar and Juan A. Rodriguez-Aguilar, editors, *Autonomous Agents and Multiagent Systems*, pages 66–83, Cham, 2017. Springer International Publishing. ISBN 978-3-319-71682-4.
2 citations in pages 88 and 89.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870. PMLR, July 2018. URL https://proceedings.mlr.press/v80/haarnoja18b.html. ISSN: 2640-3498.
4 citations in pages 12, 54, 68, and 103.

D. Halhal, G. A. Walters, D. Ouazar, and D. A. Savic. Water Network Rehabilitation with Structured Messy Genetic Algorithm. *Journal of Water Resources Planning and Management*, 123(3):137–146, 1997. doi: 10.1061/(ASCE)0733-9496(1997)123:3(137).
One citation in page 24.

Michael Hansen. Tabu search for multiobjective combinatorial optimization: TAMOCO. *Control and Cybernetics*, 29, January 2000.
One citation in page 16.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2. Publisher: Springer Science and Business Media LLC.
4 citations in pages 64, 67, 90, and 98.

Martina Hasenjäger, Bernhard Sendhoff, Toyotaka Sonoda, and Toshiyuki Arima. Three dimensional evolutionary aerodynamic design optimization with CMA-ES. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, pages 2173–2180, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1-59593-010-8. doi: 10.1145/1068009.1068366. URL https://doi.org/10.1145/1068009.1068366. event-place: Washington DC, USA.
One citation in page 24.

Conor Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa Zintgraf, Richard Dazeley, Fredrik Heintz, Enda Howley, Athirai Irissappane, Patrick Mannion, Ann Nowe, Gabriel Ramos, Marcello Restelli, Peter Vamplew, and Diederik Roijers. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36, April 2022. doi: 10.1007/s10458-022-09552-y.
6 citations in pages vii, 2, 14, 32, 62, and 97.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, pages 3207–3214, New Orleans, Louisiana, USA, February 2018. AAAI Press. ISBN 978-1-57735-800-8.
3 citations in pages 62, 74, and 75.

Matteo Hessel, Manuel Kroiss, Aidan Clark, Iurii Kemaev, John Quan, Thomas Keck, Fabio Viola, and Hado van Hasselt. Podracer architectures for scalable Reinforcement Learning, April 2021. URL http://arxiv.org/abs/2104.06272. arXiv:2104.06272 [cs].
One citation in page 101.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.
One citation in page 114.

Daniel Horn and Bernd Bischl. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, December 2016. doi: 10.1109/SSCI.2016.7850221.
One citation in page 76.

Duan Houli, Li Zhiheng, and Zhang Yi. Multiobjective Reinforcement Learning for Traffic Signal Control Using Vehicular Ad Hoc Network. *EURASIP Journal on Advances in Signal Processing*, 2010(1):1–7, December 2010. ISSN 1687-6180. doi: 10.1155/2010/724035. URL https://asp-eurasipjournals.springeropen.com/articles/10.1155/2010/724035. Number: 1 Publisher: SpringerOpen.
2 citations in pages 39 and 86.

Tianmeng Hu, Biao Luo, Chunhua Yang, and Tingwen Huang. MO-MIX: Multi-Objective Multi-Agent Cooperative Decision-Making With Deep Reinforcement Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(10):12098–12112, October 2023. ISSN 1939-3539. doi: 10.1109/TPAMI.2023.3283537. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.
2 citations in pages 38 and 97.

Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023*, 2022a.
2 citations in pages 62 and 105.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022b. ISSN 1533-7928. URL http://jmlr.org/papers/v23/21-1342.html.
3 citations in pages 54, 62, and 66.

Shengyi Huang, Quentin Gallouédec, Florian Felten, Antonin Raffin, Rousslan Fernand Julien Dossa, Yanxiao Zhao, Ryan Sullivan, Viktor Makoviychuk, Denys Makoviichuk, Mohamad H. Danesh, Cyril Roumégous, Jiayi Weng, Chufan Chen, Md Masudur Rahman, João G. M. Araújo, Guorui Quan, Daniel Tan, Timo Klein, Rujikorn Charakorn, Mark Towers, Yann Berthelot, Kinal Mehta, Dipam Chakraborty, Arjun KG, Valentin Charraut, Chang Ye, Zichen Liu, Lucas N. Alegre, Alexander Nikulin, Xiao Hu, Tianlin Liu, Jongwook Choi, and Brent Yi. Open RL Benchmark: Comprehensive Tracked Experiments for Reinforcement Learning, February 2024. URL http://arxiv.org/abs/2402.03046. arXiv:2402.03046 [cs].
10 citations in pages 52, 62, 70, 72, 73, 81, 82, 94, 101, and 121.

Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Deep reinforcement learning for swarm systems. *The Journal of Machine Learning Research*, 20(1):1966–1996, January 2019. ISSN 1532-4435.
2 citations in pages 2 and 97.

Hisao Ishibuchi and Shiori Kaige. Implementation of Simple Multiobjective Memetic Algorithms and Its Application to Knapsack Problems. *International Journal of Hybrid Intelligent Systems - IJHIS*, 1, April 2004. doi: 10.3233/HIS-2004-11-205.
One citation in page 24.

Hisao Ishibuchi, Yuji Sakane, Noritaka Tsukamoto, and Yusuke Nojima. Simultaneous Use of Different Scalarizing Functions in MOEA/D. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 519–526, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 978-1-4503-0072-8. doi: 10.1145/1830483.1830577. URL https://doi.org/10.1145/1830483.1830577. event-place: Portland, Oregon, USA.
One citation in page 21.

Joel Jang, Seungone Kim, Bill Yuchen Lin, Yizhong Wang, Jack Hessel, Luke Zettlemoyer, Hannaneh Hajishirzi, Yejin Choi, and Prithviraj Ammanabrolu. Personalized Soups: Personalized Large Language Model Alignment via Post-hoc Parameter Merging, October 2023. URL http://arxiv.org/abs/2310.11564. arXiv:2310.11564 [cs].
One citation in page 34.

Yaochu Jin and Janusz Kacprzyk, editors. *Multi-Objective Machine Learning*, volume 16 of *Studies in Computational Intelligence*. Springer, Berlin, Heidelberg, 2006. ISBN 978-3-540-30676-4 978-3-540-33019-6. doi: 10.1007/3-540-33019-4. URL http://link.springer.com/10.1007/3-540-33019-4.
One citation in page 76.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492, December 1998. ISSN 1573-2916. doi: 10.1023/A:1008306431147. URL https://doi.org/10.1023/A:1008306431147.
One citation in page 76.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, August 2021. ISSN 1476-4687. doi: 10.1038/s41586-021-03819-2. URL https://www.nature.com/articles/s41586-021-03819-2. Bandiera_abtest: a Cc_license_type: cc_by Cg_type: Nature Research Journals Number: 7873 Primary_atype: Research Publisher: Nature Publishing

Group Subject_term: Computational biophysics;Machine learning;Protein structure predictions;Structural biology Subject_term_id: computational-biophysics;machine-learning;protein-structure-predictions;structural-biology.
One citation in page 1.

Daniel Kahneman. *Thinking, fast and slow*. Penguin, London, 2012. ISBN 978-0-14-103357-0 0-14-103357-6.
2 citations in pages 13 and 24.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, August 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06419-4. URL https://www.nature.com/articles/s41586-023-06419-4. Number: 7976 Publisher: Nature Publishing Group.
3 citations in pages 1, 13, and 97.

Liangjun Ke, Qingfu Zhang, and Roberto Battiti. MOEA/D-ACO: A Multiobjective Evolutionary Algorithm Using Decomposition and AntColony. *IEEE Transactions on Cybernetics*, 43(6):1845–1859, December 2013. ISSN 2168-2275. doi: 10.1109/TSMCB.2012.2231860. Conference Name: IEEE Transactions on Cybernetics.
One citation in page 23.

N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, September 2004. doi: 10.1109/IROS.2004.1389727.
2 citations in pages 99 and 108.

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-Accelerated Parallel Game Simulators for Reinforcement Learning. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 45716–45743. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8f153093758af93861a74a1305dfdc18-Paper-Datasets_and_Benchmarks.pdf.
One citation in page 101.

Johan Källström and Fredrik Heintz. Tunable Dynamics in Agent-Based Simulation using Multi-Objective Reinforcement Learning. In *Proceedings of the 2019 Adaptive and Learning Agents Workshop (ALA), 2019*, pages 1–7. Linköping University, Faculty of Science & Engineering, 2019.
2 citations in pages 88 and 94.

Robert Tjarko Lange. gymnax: A JAX-based Reinforcement Learning Environment Library, 2022. URL http://github.com/RobertTLange/gymnax.
One citation in page 101.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. URL https://ieeexplore.ieee.org/abstract/document/726791/. Publisher: Ieee.
One citation in page 70.

Pascal Leroy, Pablo G. Morato, Jonathan Pisane, Athanasios Kolios, and Damien Ernst. IMP-MARL: a Suite of Environments for Large-scale Infrastructure Management Planning via MARL. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=q3FJk2Nvkk.
2 citations in pages 38 and 86.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research*, 18(185): 1–52, 2018. ISSN 1533-7928. URL http://jmlr.org/papers/v18/16-558.html.
2 citations in pages 76 and 116.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the Fourth International Conference on Learning Representations*, 2016. URL http://arxiv.org/abs/1509.02971. arXiv:1509.02971 [cs, stat].
One citation in page 70.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992699. URL https://doi.org/10.1007/BF00992699.
One citation in page 12.

Yiping Liu, Hisao Ishibuchi, Naoki Masuyama, and Yusuke Nojima. Adapting Reference Vectors and Scalarizing Functions by Growing Neural Gas to Handle Irregular Pareto Fronts. *IEEE Transactions on Evolutionary Computation*, 24(3):439–453, June 2020. ISSN 1941-0026. doi: 10.1109/TEVC.2019.2926151. Conference Name: IEEE Transactions on Evolutionary Computation.
One citation in page 23.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *arXiv:1706.02275 [cs]*, March 2020. URL http://arxiv.org/abs/1706.02275. arXiv: 1706.02275.
One citation in page 37.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered Policy Optimisation. *Advances in Neural Information Processing Systems*, 35: 16455–16468, December 2022a. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/688c7a82e31653e7c256c6c29fd3b438-Abstract-Conference.html.
5 citations in pages 98, 101, 103, 104, and 106.

Haoye Lu, Daniel Herman, and Yaoliang Yu. Multi-Objective Reinforcement Learning: Convexity, Stationarity and Pareto Optimality. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=TjEzIsyEsQ6.
4 citations in pages 29, 55, 67, and 68.

Junlin Lu, Patrick Mannion, and Karl Mason. A multi-objective multi-agent deep reinforcement learning approach to residential appliance scheduling. *IET Smart Grid*, 5(4):260–280, 2022b. ISSN 2515-2947. doi: 10.1049/stg2.12068. URL https://onlinelibrary.wiley.com/doi/abs/10.1049/stg2.12068. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/stg2.12068.
3 citations in pages 39, 86, and 97.

Patrick Mannion, Karl Mason, Sam Devlin, Jim Duggan, and Enda Howley. Dynamic economic emissions dispatch optimisation using multi-agent reinforcement learning. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2016)*, 2016. URL https://www.researchgate.net/profile/Patrick-Mannion-4/publication/299409768_Dynamic_Economic_Emissions_Dispatch_Optimisation_using_Multi-Agent_Reinforcement_Learning/links/56f48cb108ae7c1fda2d77af/Dynamic-Economic-Emissions-Dispatch-Optimisation-using-Multi-Agent-Reinforcement-Learning.

pdf.
One citation in page 38.

Patrick Mannion, Sam Devlin, Jim Duggan, and Enda Howley. Reward shaping for knowledge-based multi-objective multi-agent reinforcement learning. *The Knowledge Engineering Review*, 33:e23, 2018. doi: 10.1017/S0269888918000292.
One citation in page 88.

R. Marler and Jasbir Arora. The weighted sum method for multi-objective optimization: New insights. *Structural and Multidisciplinary Optimization*, 41:853–862, June 2010. doi: 10.1007/s00158-009-0460-7.
One citation in page 21.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14236. URL http://www.nature.com/articles/nature14236.
7 citations in pages 1, 12, 13, 27, 61, 70, and 81.

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based Reinforcement Learning: A Survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023. ISSN 1935-8237. doi: 10.1561/2200000086. URL http://dx.doi.org/10.1561/2200000086.
4 citations in pages 13, 31, 34, and 44.

Hossam Mossalam, Yannis M. Assael, Diederik M. Roijers, and Shimon Whiteson. Multi-Objective Deep Reinforcement Learning. *CoRR*, abs/1610.02707, 2016. URL http://arxiv.org/abs/1610.02707. arXiv: 1610.02707.
2 citations in pages 29 and 120.

J. Močkus. On bayesian methods for seeking the extremum. In G. I. Marchuk, editor, *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, Lecture Notes in Computer Science, pages 400–404, Berlin, Heidelberg, 1975. Springer. ISBN 978-3-540-37497-8. doi: 10.1007/3-540-07165-2_55.
One citation in page 76.

Tadahiko Murata, Hisao Ishibuchi, and Mitsuo Gen. Specification of Genetic Search Directions in Cellular Multi-objective Genetic Algorithms. In Eckart Zitzler, Lothar Thiele, Kalyanmoy Deb, Carlos Artemio Coello Coello, and David Corne, editors, *Evolutionary Multi-Criterion Optimization*, Lecture Notes in Computer Science, pages 82–95, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-44719-1. doi: 10.1007/3-540-44719-9_6.
One citation in page 23.

Sriraam Natarajan and Prasad Tadepalli. *Dynamic preferences in multi-criteria reinforcement learning*. Association for Computing Machinery, January 2005. doi: 10.1145/1102351.1102427. Pages: 608.
2 citations in pages 33 and 93.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning*, volume 99, pages 278–287, 1999.
One citation in page 99.

Robert Ojstersek, Miran Brezocnik, and Borut Buchmeister. Multi-objective optimization of production scheduling with evolutionary computation: A review. *International Journal of Industrial Engineering*

Computations, 11, January 2020. doi: 10.5267/j.ijiec.2020.1.003.
One citation in page 24.

Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Springer International Publishing, Cham, 2016. ISBN 978-3-319-28927-4 978-3-319-28929-8. doi: 10.1007/978-3-319-28929-8. URL http://link.springer.com/10.1007/978-3-319-28929-8.
One citation in page 36.

Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *J. Artif. Int. Res.*, 32(1):289–353, May 2008. ISSN 1076-9757. Place: El Segundo, CA, USA Publisher: AI Access Foundation.
One citation in page 37.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.
2 citations in pages 1 and 13.

Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. Automated Reinforcement Learning (AutoRL): A Survey and Open Problems. *Journal of Artificial Intelligence Research*, 74, September 2022. ISSN 1076-9757. doi: 10.1613/jair.1.13596. URL https://dl.acm.org/doi/10.1613/jair.1.13596.
3 citations in pages 51, 75, and 76.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.
One citation in page 67.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical Design in Reinforcement Learning, April 2023. URL http://arxiv.org/abs/2304.01315. arXiv:2304.01315 [cs].
5 citations in pages 3, 61, 62, 74, and 86.

Yutao Qi, Xiaoliang Ma, Fang Liu, Licheng Jiao, Jianyong Sun, and Jianshe Wu. MOEA/D with adaptive weight adjustment. *Evolutionary computation*, 22, June 2013. doi: 10.1162/EVCO_a_00109.
One citation in page 22.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.
2 citations in pages 62 and 66.

Gabriel de O Ramos, Roxana Rădulescu, Ann Nowé, and Anderson R Tavares. Toll-based learning for minimising congestion under heterogeneous preferences. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1098–1106, 2020.
One citation in page 88.

Alexandre Ramé, Guillaume Couairon, Mustafa Shukor, Corentin Dancette, Jean-Baptiste Gaya, Laure Soulier, and Matthieu Cord. Rewarded soups: towards Pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards, October 2023. URL http://arxiv.org/abs/2306.04488. arXiv:2306.04488 [cs].
One citation in page 34.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020. URL https://www.jmlr.org/papers/v21/20-081.html.
2 citations in pages 37 and 38.

Mathieu Reymond and Ann Nowe. Pareto-DQN: Approximating the Pareto front in complex multi-objective decision problems. In *Proceedings of the Adaptive and Learning Agents Workshop 2019 (ALA-19) at AAMAS*, May 2019. URL https://ala2019.vub.ac.be.
One citation in page 27.

Mathieu Reymond, Eugenio Bargiacchi, and Ann Nowe. Pareto Conditioned Networks. In *The 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 1110–1118. IFAAMAS, May 2022. URL https://aamas2022-conference.auckland.ac.nz.
3 citations in pages 67, 68, and 95.

Diederik Roijers, Shimon Whiteson, and Frans Oliehoek. Computing Convex Coverage Sets for Faster Multi-objective Coordination. *Journal of Artificial Intelligence Research*, 52:399–443, March 2015a. doi: 10.1613/jair.4550.
2 citations in pages 88 and 120.

Diederik Roijers, Denis Steckelmacher, and Ann Nowe. Multi-objective Reinforcement Learning for the Expected Utility of the Return. In *Proceedings of the ALA workshop at ICML/AAMAS/IJCAI 2018*, July 2018.
5 citations in pages 31, 32, 57, 67, and 68.

Diederik M. Roijers and Shimon Whiteson. Multi-Objective Decision Making. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 11(1):1–129, April 2017. ISSN 1939-4608. doi: 10.2200/S00765ED1V01Y201704AIM034. URL https://www.morganclaypool.com/doi/10.2200/S00765ED1V01Y201704AIM034. Publisher: Morgan & Claypool Publishers.
9 citations in pages 2, 25, 28, 38, 67, 68, 69, 93, and 119.

Diederik M. Roijers, Shimon Whiteson, and Frans A. Oliehoek. Point-Based Planning for Multi-Objective POMDPs. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1666–1672. AAAI Press, 2015b. ISBN 978-1-57735-738-4. Place: Buenos Aires, Argentina.
3 citations in pages 33, 58, and 115.

Diederik M Roijers, Willem Röpke, Ann Nowé, and Roxana Rădulescu. On Following Pareto-Optimal Policies in Multi-Objective Planning and Reinforcement Learning. In *Proceedings of the 1st Multi-Objective Decision Making Workshop (MODeM)*, page 8, 2021.
One citation in page 27.

Diederik Marijn Roijers. *Multi-Objective Decision-Theoretic Planning*. PhD Thesis, University of Amsterdam, 2016.
One citation in page 88.

Diederik Marijn Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A Survey of Multi-Objective Sequential Decision-Making. *Journal of Artificial Intelligence Research*, 48:67–113, October 2013. ISSN 1076-9757. doi: 10.1613/jair.3987. URL http://arxiv.org/abs/1402.0590. arXiv: 1402.0590.
One citation in page 3.

Manuela Ruiz-Montiel, Lawrence Mandow, and José-Luis Pérez-de-la Cruz. A temporal difference method for multi-objective reinforcement learning. *Neurocomputing*, 263:15–25, November 2017. ISSN 0925-2312. doi: 10.1016/j.neucom.2016.10.100. URL https://www.sciencedirect.com/science/article/pii/S0925231217310998.
4 citations in pages 26, 43, 44, and 46.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.
One citation in page 1.

Jean-Charles Régin, Mohamed Rezgui, and Arnaud Malapert. Embarrassingly Parallel Search. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 596–610, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-40627-0. doi: 10.1007/978-3-642-40627-0_45.
2 citations in pages 98 and 101.

Willem Röpke, Conor F. Hayes, Patrick Mannion, Enda Howley, Ann Nowé, and Diederik M. Roijers. Distributional Multi-Objective Decision Making. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 5711–5719, Macau, SAR China, August 2023. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-1-956792-03-4. doi: 10.24963/ijcai.2023/634. URL https://www.ijcai.org/proceedings/2023/634.
One citation in page 32.

Willem Röpke, Mathieu Reymond, Patrick Mannion, Diederik M. Roijers, Ann Nowé, and Roxana Rădulescu. Divide and Conquer: Provably Unveiling the Pareto Front with Multi-Objective Reinforcement Learning, February 2024. URL http://arxiv.org/abs/2402.07182. arXiv:2402.07182 [cs].
One citation in page 115.

Roxana Rădulescu, Patrick Mannion, Diederik M. Roijers, and Ann Nowé. Multi-objective multi-agent decision making: a utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems*, 34(1):10, April 2020. ISSN 1573-7454. doi: 10.1007/s10458-019-09433-x. URL https://doi.org/10.1007/s10458-019-09433-x.
12 citations in pages vii, 2, 3, 35, 36, 38, 85, 88, 92, 94, 97, and 116.

Alejandro Santiago, Héctor Joaquín Fraire Huacuja, Bernabé Dorronsoro, Johnatan E. Pecero, Claudia Gómez Santillan, Juan Javier González Barbosa, and José Carlos Soto Monterrubio. A Survey of Decomposition Methods for Multi-objective Optimization. In Oscar Castillo, Patricia Melin, Witold Pedrycz, and Janusz Kacprzyk, editors, *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, Studies in Computational Intelligence, pages 453–465. Springer International Publishing, Cham, 2014. ISBN 978-3-319-05170-3. doi: 10.1007/978-3-319-05170-3_31. URL https://doi.org/10.1007/978-3-319-05170-3_31.
One citation in page 20.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. In Yoshua Bengio and Yann LeCun, editors, *Proceedings of the 4th International Conference on Learning Representations,*

(ICLR 2016), 2016. URL http://arxiv.org/abs/1511.05952.
2 citations in pages 13 and 30.

CA Schroeder de Witt. *Coordination and communication in deep multi-agent reinforcement learning.* PhD Thesis, University of Oxford, 2021.
4 citations in pages 36, 37, 38, and 93.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017.
8 citations in pages 12, 27, 55, 62, 68, 71, 99, and 103.

Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-Free Replicated Data Types. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 6976, pages 386–400. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-24549-7 978-3-642-24550-3. doi: 10.1007/978-3-642-24550-3_29. URL http://link.springer.com/10.1007/978-3-642-24550-3_29. Series Title: Lecture Notes in Computer Science.
One citation in page 115.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 7587 Primary_atype: Research Publisher: Nature Publishing Group Subject_term: Computational science;Computer science;Reward Subject_term_id: computational-science;computer-science;reward.
4 citations in pages 1, 13, 61, and 97.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning series. A Bradford Book, Cambridge, MA, USA, 2 edition, November 2018. ISBN 978-0-262-03924-6.
7 citations in pages 1, 9, 10, 12, 13, 61, and 75.

Jun Jet Tai, Jim Wong, Mauro Innocente, Nadjim Horri, James Brusey, and Swee King Phang. PyFlyt – UAV Simulation Environments for Reinforcement Learning Research, April 2023. URL http://arxiv.org/abs/2304.01305. arXiv:2304.01305 [cs].
2 citations in pages 99 and 108.

El-Ghazali Talbi. *Metaheuristics: From Design to Implementation.* Wiley Publishing, 2009. ISBN 0-470-27858-7.
8 citations in pages vii, 15, 16, 18, 19, 20, 24, and 42.

Ming Tan. Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents. In Paul E. Utgoff, editor, *Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993*, pages 330–337. Morgan Kaufmann, 1993. doi: 10.1016/b978-1-55860-307-3.50049-6. URL https://doi.org/10.1016/b978-1-55860-307-3.50049-6.
One citation in page 36.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, Niall Williams, Yashas Lokesh, and Praveen Ravi. PettingZoo: Gym for Multi-Agent Reinforcement Learning. In *Advances in Neural Information*

Processing Systems, volume 34, pages 15032–15043. Curran Associates, Inc., 2021.
5 citations in pages 63, 85, 86, 89, and 90.

J. K. Terry, Benjamin Black, and Ananth Hari. SuperSuit: Simple Microwrappers for Reinforcement Learning Environments, August 2020. URL http://arxiv.org/abs/2008.08932. arXiv:2008.08932 [cs].
One citation in page 91.

Luiz A Thomasini, Lucas N Alegre, Gabriel de O Ramos, and Ana LC Bazzan. RouteChoiceEnv: a Route Choice Library for Multiagent Reinforcement Learning. In *Adaptive and Learning Agents Workshop (ALA 2023) at AAMAS, London, UK, 2023*, 2023.
2 citations in pages 88 and 89.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, October 2012. doi: 10.1109/IROS.2012.6386109. ISSN: 2153-0866.
One citation in page 52.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL https://zenodo.org/record/8127025.
3 citations in pages 62, 63, and 86.

Brendan D. Tracey, Andrea Michi, Yuri Chervonyi, Ian Davies, Cosmin Paduraru, Nevena Lazic, Federico Felici, Timo Ewalds, Craig Donner, Cristian Galperti, Jonas Buchli, Michael Neunert, Andrea Huber, Jonathan Evens, Paula Kurylowicz, Daniel J. Mankowitz, and Martin Riedmiller. Towards practical reinforcement learning for tokamak magnetic control. *Fusion Engineering and Design*, 200:114161, March 2024. ISSN 0920-3796. doi: 10.1016/j.fusengdes.2024.114161. URL https://www.sciencedirect.com/science/article/pii/S0920379624000140.
2 citations in pages 1 and 13.

Peter Vamplew, Richard Dazeley, Ewan Barker, and Andrei Kelarev. Constructing Stochastic Mixture Policies for Episodic Multiobjective Reinforcement Learning Tasks. In Ann Nicholson and Xiaodong Li, editors, *AI 2009: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 340–349, Berlin, Heidelberg, 2009. Springer. ISBN 978-3-642-10439-8. doi: 10.1007/978-3-642-10439-8_35.
One citation in page 31.

Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84(1):51–80, July 2011. ISSN 1573-0565. doi: 10.1007/s10994-010-5232-5. URL https://doi.org/10.1007/s10994-010-5232-5.
3 citations in pages 46, 52, and 65.

Peter Vamplew, Richard Dazeley, and Cameron Foale. Softmax exploration strategies for multiobjective reinforcement learning. *Neurocomputing*, 263:74–86, November 2017a. ISSN 09252312. doi: 10.1016/j.neucom.2016.09.141. URL https://linkinghub.elsevier.com/retrieve/pii/S0925231217310974.
2 citations in pages 13 and 42.

Peter Vamplew, Rustam Issabekov, Richard Dazeley, Cameron Foale, Adam Berry, Tim Moore, and Douglas Creighton. Steering approaches to Pareto-optimal multiobjective reinforcement learning. *Neurocomputing*, 263:26–38, November 2017b. ISSN 0925-2312. doi: 10.1016/j.neucom.2016.08.152. URL https://www.

sciencedirect.com/science/article/pii/S0925231217311013.
One citation in page 114.

Peter Vamplew, Dean Webb, Luisa Zintgraf, Diederik Roijers, Richard Dazeley, Rustam Issabekov, and Evan Dekker. MORL-Glue: A Benchmark Suite for Multi-Objective Reinforcement Learning. In Bart Verheij and Marco Wiering, editors, *Proceedings of the 29th Benelux Conference on Artificial Intelligence*. Benelux Association for Artificial Intelligence (BNVKI-AIABN), August 2017c. URL http://bnaic2017.ai.rug.nl/,https://bnaic2017.ai.rug.nl.
One citation in page 62.

Peter Vamplew, Benjamin J. Smith, Johan Källström, Gabriel Ramos, Roxana Rădulescu, Diederik M. Roijers, Conor F. Hayes, Fredrik Heintz, Patrick Mannion, Pieter J. K. Libin, Richard Dazeley, and Cameron Foale. Scalar reward is not enough: a response to Silver, Singh, Precup and Sutton (2021). *Autonomous Agents and Multi-Agent Systems*, 36(2):41, July 2022. ISSN 1573-7454. doi: 10.1007/s10458-022-09575-5. URL https://doi.org/10.1007/s10458-022-09575-5.
3 citations in pages 2, 14, and 34.

Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512, 2014. Publisher: JMLR. org.
7 citations in pages x, 26, 43, 44, 46, 67, and 68.

Kristof Van Moffaert, Madalina M. Drugan, and Ann Nowe. Scalarized multi-objective reinforcement learning: Novel design techniques. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 191–199, Singapore, Singapore, April 2013. IEEE. ISBN 978-1-4673-5925-2. doi: 10.1109/ADPRL.2013.6615007. URL http://ieeexplore.ieee.org/document/6615007/.
2 citations in pages 57 and 67.

Sébastien Varrette, Pascal Bouvry, Hyacinthe Cartiaux, and Fotis Georgatos. Management of an academic HPC cluster: The UL experience. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*, pages 959–967. IEEE, 2014.
4 citations in pages 52, 71, 79, and 94.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.
One citation in page 114.

Christopher Watkins. *Learning From Delayed Rewards*. PhD thesis, King's College, Oxford, January 1989.
2 citations in pages 28 and 67.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.
One citation in page 12.

Marco A. Wiering, Maikel Withagen, and Madalina M Drugan. Model-based multi-objective reinforcement learning. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 1–6, Orlando, FL, USA, December 2014. IEEE. ISBN 978-1-4799-4552-8. doi: 10.1109/ADPRL.2014.7010622. URL http://ieeexplore.ieee.org/document/7010622/.
One citation in page 34.

Barry Wilkinson and Michael Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Prentice-Hall, Inc., USA, 2nd edition, 2005. ISBN 0-13-671710-1.
2 citations in pages 98 and 101.

D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997. ISSN 1941-0026. doi: 10.1109/4235.585893. URL https://ieeexplore.ieee.org/document/585893. Conference Name: IEEE Transactions on Evolutionary Computation.
One citation in page 51.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S. Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *Proceedings of the 39th International Conference on Machine Learning*, pages 23965–23998. PMLR, June 2022. URL https://proceedings.mlr.press/v162/wortsman22a.html. ISSN: 2640-3498.
One citation in page 115.

Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, February 2022. ISSN 1476-4687. doi: 10.1038/s41586-021-04357-7. URL https://www.nature.com/articles/s41586-021-04357-7. Number: 7896 Publisher: Nature Publishing Group.
One citation in page 13.

Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-Guided Multi-Objective Reinforcement Learning for Continuous Robot Control. In *Proceedings of the 37th International Conference on Machine Learning*, pages 10607–10616. PMLR, November 2020a. URL https://proceedings.mlr.press/v119/xu20h.html. ISSN: 2640-3498.
9 citations in pages 17, 29, 55, 62, 65, 67, 68, 119, and 120.

Qian Xu, Zhanqi Xu, and Tao Ma. A Survey of Multiobjective Evolutionary Algorithms Based on Decomposition: Variants, Challenges and Future Directions. *IEEE Access*, 8:41588–41614, 2020b. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2973670. Conference Name: IEEE Access.
One citation in page 20.

Runzhe Yang, Xingyuan Sun, and Karthik Narasimhan. A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/hash/4a46fbfca3f1465a27b210f4bdfe6ab3-Abstract.html.
9 citations in pages 30, 31, 33, 62, 67, 68, 78, 81, and 120.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=YVXaxB6L2Pl.
5 citations in pages 33, 37, 92, 93, and 103.

Qingfu Zhang and Hui Li. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, December 2007. ISSN 1941-0026. doi:

10.1109/TEVC.2007.892759. Conference Name: IEEE Transactions on Evolutionary Computation.
3 citations in pages 16, 23, and 24.

Yanyun Zhang, Guanyu Chen, Li Cheng, Quanyu Wang, and Qi Li. Methods to balance the exploration and exploitation in Differential Evolution from different scales: A survey. *Neurocomputing*, 561:126899, December 2023. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.126899. URL https://www.sciencedirect.com/science/article/pii/S0925231223010226.
One citation in page 13.

Stephan Zheng, Alexander Trott, Sunil Srinivasa, David C. Parkes, and Richard Socher. The AI Economist: Taxation policy design via two-level deep multiagent reinforcement learning. *Science Advances*, 8(18):eabk2607, 2022. doi: 10.1126/sciadv.abk2607. URL https://www.science.org/doi/abs/10.1126/sciadv.abk2607. _eprint: https://www.science.org/doi/pdf/10.1126/sciadv.abk2607.
One citation in page 39.

Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N. Zare, and Patrick Riley. Optimization of Molecules via Deep Reinforcement Learning. *Scientific Reports*, 9(1):10752, July 2019. ISSN 2045-2322. doi: 10.1038/s41598-019-47148-x. URL https://www.nature.com/articles/s41598-019-47148-x. Number: 1 Publisher: Nature Publishing Group.
One citation in page 34.

Baiting Zhu, Meihua Dang, and Aditya Grover. Pareto-Efficient Decision Agents for Offline Multi-Objective Reinforcement Learning. In *3rd Offline RL Workshop: Offline RL as a "Launchpad"*, 2022. URL https://openreview.net/forum?id=tSJ4ykkddy.
One citation in page 63.

Luisa M Zintgraf, Timon V Kanters, Diederik M Roijers, Frans A Oliehoek, and Philipp Beau. Quality Assessment of MORL Algorithms: A Utility-Based Approach. In *Benelearn 2015: Proceedings of the 24th Annual Machine Learning Conference of Belgium and the Netherlands*, 2015.
One citation in page 18.

Luisa M Zintgraf, Diederik M Roijers, Sjoerd Linders, Catholijn M Jonker, and Ann Nowé. Ordered Preference Elicitation Strategies for Supporting Multi-Objective Decision Making. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*, 2018.
One citation in page 114.

E. Zitzler. Evolutionary algorithms for multiobjective optimization: methods and applications. In *Ph.D. Dissertation. ETH Zurich, Switzerland.*, December 1999.
One citation in page 17.

Eckart Zitzler and Simon Künzli. Indicator-Based Selection in Multiobjective Search. In Xin Yao, Edmund K. Burke, José A. Lozano, Jim Smith, Juan Julián Merelo-Guervós, John A. Bullinaria, Jonathan E. Rowe, Peter Tiňo, Ata Kabán, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, Lecture Notes in Computer Science, pages 832–842, Berlin, Heidelberg, 2004. Springer. ISBN 978-3-540-30217-9. doi: 10.1007/978-3-540-30217-9_84.
One citation in page 19.