Faculty of Science and Bio-Engineering Sciences
↪ Department of Computer Science
↪ Artificial Intelligence Laboratory

On incorporating prior knowledge about the decision maker in multi-objective reinforcement learning

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

Mathieu Reymond

Promotor: Prof. Dr. Ann Nowé (Vrije Universiteit Brussel)
Co-promotor: Dr. Diederik M. Roijers (Vrije Universiteit Brussel)

Printed by
Crazy Copy Center Productions
VUB Pleinlaan 2, 1050 Brussel
Tel : +32 2 629 33 44
crazycopy@vub.be
www.crazycopy.be

ISBN : 9789464443738
NUR CODE : 984
THEMA : UYQM

Abstract

Decision makers acting in real-world problems often have to take into account multiple objectives. When maximizing one objective comes at the cost of another, the objectives are in conflict, and the decision maker must find a compromise between them. The optimal trade-offs might differ on a case-by-case basis, as they depend on the preferences, or utility, of the decision maker.

In this dissertation, we investigate utility-based approaches, that take into account prior knowledge over the decision maker's utility, to learn his or her preferred trade-off. By making efficient usage of this prior knowledge, we can swiftly discard undesirable trade-offs, significantly narrowing down the search space for the optimal solution, and thus enhancing the speed at which we find the desired trade-off.

We analyze different scenarios, depending on the amount of prior knowledge available. First, we consider that the utility is known a priori, and propose a novel multi-objective reinforcement learning algorithm that optimizes directly on said utility. We show that, by explicitly considering multiple objectives, we learn the optimal trade-off in a more stable and efficient manner than using single-objective solvers. Second, we consider the interactive scenario, with only partial prior knowledge over the utility, but where we can query the decision maker to learn about its preferences. We show that we can optimize the timing of our queries during the learning process to maximally improve our chances of learning the preferred trade-off. Third, we assume no prior knowledge over the utility. Learning a single, but conditional solution on preferences allows us to reuse the samples learned for different trade-offs, thus improving the efficiency of the search. By generalizing our solution to all possible preferences, we can learn any possible trade-off, such that the decision maker can choose its preferred solution a posteriori.

Throughout our research, we primarily focus on sequential decision-making problems, where a solution is found after having taken a sequence of actions. We demonstrate our findings for unknown utility functions on a real-world use-case, epidemic outbreaks. By reducing the number of social contacts at key points in time, we can control the spread of the epidemic. We learn a variety of optimal strategies that balance between the hospitalizations due to the outbreak and the social contact reduction, which can support the decision maker in taking an informed decision, knowing the available alternatives.

Samenvatting

Besluitvormers moeten vaak rekening houden met meerdere doelstellingen. Wanneer het maximaliseren van één doelstelling ten koste gaat van een andere, zijn de doelstellingen conflicterend, en moet de besluitvormer een compromisoplossing vinden. De optimale afwegingen kunnen per geval verschillen, omdat ze afhangen van de voorkeuren, of het nut, van de beslisser.

In dit proefschrift onderzoeken we nutsgebaseerde benaderingen, die rekening houden met voorkennis over het nut van de beslisser, om zo zijn of haar voorkeursafweging te leren kennen. Door efficiënt gebruik te maken van deze voorkennis kunnen we ongewenste afwegingen snel verwerpen, waardoor de zoekruimte naar de optimale oplossing aanzienlijk kleiner wordt en we dus sneller de gewenste afweging vinden.

Wij analyseren verschillende scenario's, afhankelijk van de beschikbare voorkennis. Ten eerste nemen we aan dat het nut a priori bekend is, en stellen we een nieuw multi-objectief reinforcement learning algoritme voor dat rechtstreeks op dat nut optimaliseert. We tonen aan dat, door expliciet rekening te houden met meerdere objectieven, we de optimale afweging leren op een stabielere en efficiëntere manier dan met single-objective algoritmen. Ten tweede beschouwen wij het interactieve scenario, met slechts gedeeltelijke voorkennis over het nut, maar waarbij wij met de beslisser kunnen interageren om zijn voorkeuren te leren kennen. We kunnen hierbij de timing van onze vragen optimaliseren om de probabiliteit op het leren van de gewenste compromis te maximaliseren. Ten derde veronderstellen wij geen voorkennis over het nut. Het leren van een enkele, maar voorwaardelijke oplossing op voorkeuren stelt ons in staat de geleerde data te hergebruiken voor verschillende afwegingen, waardoor de efficiëntie van het zoekproces toeneemt. Door onze oplossing te generaliseren naar alle mogelijke voorkeuren, kunnen we elke mogelijke afweging leren, zodat de beslisser a posteriori zijn voorkeursoplossing kan kiezen.

In ons onderzoek richten wij ons voornamelijk op sequentiële besluitvormingsproblemen, waarbij een oplossing wordt gevonden na het nemen van een reeks handelingen. Wij demonstreren onze bevindingen voor onbekende nutsfuncties op een reëel praktijkgeval, namelijk epidemische uitbraken. Door het aantal sociale contacten op kritische momenten te beperken, kunnen we de verspreiding van de epidemie beheersen. Wij leren een verscheidenheid aan optimale strategieën die een evenwicht vinden tussen de ziekenhuisopnames ten gevolge van de uitbraak en de vermindering van het aantal sociale contacten, wat de besluitvormer kan helpen bij het nemen van een geïnformeerde beslissing, met kennis van de beschikbare alternatieven.

Acknowledgements

As I look back and reflect on the past 6 years that I spent on this doctoral dissertation, I realize that I would not have been able to complete it without the help and involvement of a great number of people. I would like to take the opportunity to thank all of them, although, I have to say, words fall short.

First and foremost, I would like to express my deepest gratitude to prof. dr. Ann Nowé, my promotor, and dr. Diederik M. Roijers, my co-promotor. Ann, thank you for giving me the opportunity to join the AI Lab and to start a doctoral dissertation, it has made me realize at which point I enjoy research. Thank you for your continuous guidance and support, and for providing such a positive environment. Diederik, my most sincere thanks for introducing me to the world of multi-objective optimization. With your help and guidance, it has allowed me to further refine my research into a topic I feel really passionate about.

I am also indebted to the members of my doctoral exam commission, prof. dr. Peter Vamplew, prof. dr. Frederik Heintz, prof. dr. Sam Verboven, prof. dr. Elisa Gonzalez Boix, and prof. dr. Geraint A. Wiggins, the chairman of the commission. Thank you for the time and effort put into assessing my dissertation, as well as the discussion, constructive feedback, and valuable suggestions, which have significantly enhanced the quality of this work.

I believe I would not have been able to enjoy research as much as I do had it not been for the constructive and encouraging atmosphere of the AI Lab and all its members. Arno and Youri, my office would have been much more dull without you, even though it was strategically located next to the coffee machine. Felipe, Pieter and Timo, thank you, f(l)oreal, for making me feel right at home at the start of my PhD, with your good humor and unending jokes. Kirk, thank you for all your knowledge about deep learning, and for always smuggling homemade Greek delicacies. Thank you, Denis, Hélène and Roxana, for all the pertinent discussions and sharing your knowledge with me. Florent and Raphaël, it has been my greatest pleasure to have you as post-COVID office-mates and turn it into a French-speaking bastion, to Willem's great dismay. Much love to you, Willem. Eugenio, I think your contagious laugh will forever echo in my mind. Thank you for all the games, C++ trickery and collaborations. I would also like to express my gratitude to Brigitte and Frederik, for their administrative and technical support, Bart, for inspiring me while I was his teaching assistant, and all the other members of the lab, for the lab drinks and good mood.

Next to the lab members, I have also been blessed with great friends that I could always count on and would keep me in high spirits in stressful times. In particular, I have spent some of the best moments of these past few years with my flatmates, Alain, Amaury, Béné and Tanguy. I will cherish all the trips, holidays, and evenings spent together.

I am also deeply thankful to my parents, for providing me with the opportunities and freedom to pursue the studies of my choice, believing in me when I struggled, and raising me into who I am today.

Finally, thank you, Margaux, not only for your unwavering support, but also simply for being who you are, you have put immeasurable joy in my heart.

Contents

Contents **1**

1 Introduction **5**
1.1 Introduction . . . 5
1.2 Motivating scenarios for multi-objective approaches . . . 7
1.3 Research question and contributions . . . 9
1.3.1 Known, non-linear utility functions . . . 10
1.3.2 Partially known utility function, with interactive learning . . . 10
1.3.3 Unknown, non-linear utility function . . . 10
1.3.4 Learning mitigation policies for epidemic outbreaks . . . 11
1.4 Organization of this dissertation . . . 11

2 Background **13**
2.1 Multi-objective reinforcement learning . . . 13
2.2 Multi-objective Markov decision processes . . . 14
2.3 Policies . . . 14
2.4 Utility function . . . 15
2.5 Optimality criterion . . . 17
2.5.1 The special case of linear scalarization . . . 18
2.6 Coverage sets . . . 19
2.6.1 Linear utility functions under SER and ESR . . . 20
2.6.2 Monotonically increasing utility functions under SER . . . 21
2.6.3 The impact of stochastic policies on the coverage set . . . 21
2.7 The utility-based perspective . . . 23
2.8 Performance metrics for multi-objective optimization . . . 24
2.8.1 Hypervolume . . . 24
2.8.2 Expected Utility Metric . . . 25
2.8.3 Expected Utility Loss . . . 25
2.8.4 Maximal Utility Loss . . . 25
2.8.5 The ε-indicator . . . 26
2.8.6 The $\varepsilon - mean$-indicator . . . 26

3 Incorporating non-linear utility functions to learn the optimal policy under ESR **29**
3.1 The challenge of the ESR optimization criterion . . . 30

3.2 Using accrued rewards to compute the utility . . . 31
3.2.1 Markov property for MOMDPs with accrued rewards . . . 31
3.3 Optimizing the utility using single-objective methods . . . 31
3.3.1 Mapping function for linear scalarization functions . . . 32
3.4 Policy gradient for multi-objective reward functions . . . 33
3.5 Using a distributional critic to estimate the future returns . . . 35
3.6 Multi-Objective Categorical Actor-Critic . . . 36
3.6.1 Multi-objective policy gradient with future return estimates . . . 37
3.6.2 Incorporating a distributional critic . . . 39
3.6.3 Corollary: A multi-objective actor for SER . . . 41
3.7 Ablation study . . . 43
3.7.1 Using a distributional critic . . . 43
3.7.2 Conditioning on accrued rewards . . . 44
3.7.3 Illustration on general monotonically increasing utility functions . 46
3.8 Benchmark environments . . . 48
3.8.1 Deep-Sea-Treasure . . . 48
3.8.2 Minecart . . . 49
3.9 Experiments . . . 50
3.9.1 Deep-Sea-Treasure . . . 51
3.9.2 Minecart . . . 53
3.10 Related work . . . 55
3.11 Discussion . . . 56

4 Improving our knowledge about the utility function 59
4.1 Introduction . . . 60
4.2 Multi-armed bandits . . . 61
4.3 Algorithms for best-arm identification . . . 62
4.3.1 Round robin . . . 62
4.3.2 Upper Confidence Bound . . . 62
4.3.3 Top-two Thompson Sampling . . . 63
4.4 Multi-objective multi-armed bandits . . . 65
4.4.1 Multi-objective top-two Thompson sampling . . . 67
4.4.2 Belief distribution of the utility function . . . 67
4.4.3 Comparison between Bayesian logistic regression and particle filtering . . . 69
4.4.4 Selecting queries for the decision maker . . . 71
4.4.5 Different query timings for multi-objective Top-two Thompson sampling . . . 72
4.4.6 Experiments . . . 72
4.5 POMCP for query optimization . . . 76
4.5.1 Simulating rollouts . . . 77
4.5.2 Transition model of MOMABs for simulated rollouts . . . 78
4.5.3 Aggregating belief-nodes together . . . 79
4.5.4 Evaluating rollouts . . . 80
4.5.5 Experiments . . . 81
4.6 Related work . . . 82
4.7 Discussion . . . 83

5 Learning Pareto efficient policies for non-linear utility functions 85
5.1 Introduction . . . 85
5.2 Reward Conditioned Policies . . . 87
5.3 Pareto Conditioned Networks . . . 87
5.3.1 Building the dataset . . . 88
5.3.2 Training the Network . . . 88
5.3.3 Policy Exploration . . . 89
5.3.4 Updating the Dataset . . . 90
5.4 Experiments . . . 91
5.4.1 Deep-Sea-Treasure . . . 93
5.4.2 Minecart . . . 94
5.4.3 Crossroad . . . 94
5.5 Scaling Up the Objective-Space . . . 96
5.6 Related work . . . 98
5.7 Discussion . . . 100

6 Use-case: exploring the Pareto front of multi-objective COVID-19 mitigation policies using reinforcement learning 101
6.1 Introduction . . . 102
6.2 COVID-19 model and the MOBelCov MOMDP . . . 103
6.2.1 Stochastic compartment model for SARS-CoV-2 . . . 103
6.2.2 Interventions strategies . . . 106
6.2.3 A MOMDP for the compartment model . . . 106
6.3 Pareto Conditioned Networks for MOBelCov . . . 109
6.3.1 Training the network for continuous actions . . . 109
6.3.2 Coping with stochastic transitions . . . 110
6.4 Experiments . . . 110
6.4.1 Learned coverage set . . . 111
6.4.2 Impact of budgets on the coverage set . . . 112
6.4.3 R_{ARH} versus on R_{ARI} . . . 113
6.4.4 Robustness of policy executions . . . 116
6.5 Related work . . . 117
6.6 Discussion . . . 117

7 Conclusions and future work 119
7.1 Future work . . . 120
7.1.1 Conditional reinforcement learning . . . 123
7.1.2 Distributional reinforcement learning . . . 123
7.1.3 Model-based reinforcement learning . . . 124

Bibliography 127

Publications 145

Appendix A Appendix for Chapter 3 149
A.1 Additional results . . . 149
A.2 Hyperparameters . . . 149

Appendix B Appendix for Chapter 6 **153**
B.1 Stochastic Compartmental Model 153
B.1.1 Modelling interventions 155
B.2 Additional results 155
B.2.1 Comparison of coverage sets learned on ODE and Binomial models 155
B.2.2 Comparison of coverage sets learned on $\mathbf{R}_{\text{ARH}}$ and $\mathbf{R}_{\text{ARI}}$ 155
B.2.3 Experiment parameters 157
B.2.4 Neural network architecture 157
B.2.5 Policy executions 158

Chapter 1

Introduction

1.1 Introduction 5
1.2 Motivating scenarios for multi-objective approaches 7
1.3 Research question and contributions 9
1.3.1 Known, non-linear utility functions 10
1.3.2 Partially known utility function, with interactive learning 10
1.3.3 Unknown, non-linear utility function 10
1.3.4 Learning mitigation policies for epidemic outbreaks 11
1.4 Organization of this dissertation 11

1.1 Introduction

Many real-world decision problems are sequential in nature. Say we want to provide electricity to a region using hydroelectric power. This power is produced through a set of dams situated at strategic locations of water reservoirs. By regulating the flow of water that passes by the dams through the control of the dams' gates, we produce electricity. Our aim is to produce enough electricity such that the demand for the region is met. However, the amount of water we release now has an impact on how much we can release later on. Maybe, at a later point in time, there will not be enough water left in the reservoirs to meet the demand. Thus, our dam controller should not optimize the instantaneous opening of its gates as to meet the current demand. Instead, it should optimize the sequence of openings over time such that the demand is always met (Castelletti et al., 2013; Pianosi et al., 2013).

Another example of such a sequential decision problem that garnered world-wide attention the past three years concerns the mitigation of the COVID pandemic. Governments imposed a series of decisions to root out the epidemic, mostly in the form of social restrictions such as lockdowns or travel constraints (Willem et al., 2021). It is the whole sequence

of decisions over time that led to a more or less successful reduction of the number of infections of the pandemic.

Different approaches exist to solve such sequential decision problems. In this dissertation, we focus on one set of approaches called reinforcement learning (RL) (Sutton & Barto, 2018). At its core, we can see RL as a learning process that tries out different strategies, reinforcing the decisions that lead to desirable results and trying out alternatives otherwise. We typically say that RL searches for the optimal strategy through trial and error.

In both the abovementioned examples it would be helpful to have a single, clear objective for which RL can optimize: the electricity demand should always be met, and the pandemic should be rooted out as quickly as possible. Considering this, a straightforward solution to root out the COVID pandemic would have been to lock everyone into their homes until not a single infection remained. Sadly, doing so would have led to disastrous economic consequences and would have considerably impacted the health and psychological well-being of a number of individuals. We can understand why this strategy has not been applied by our governments. In reality, they had to consider additional objectives such as economic impact and mental well-being next to the infection mitigation. Analogously, regulating the water flow of water reservoirs has significant socio-economic impacts for the region. The release of water downstream is necessary to meet downstream users' agricultural needs (Castelletti et al., 2013; Pianosi et al., 2013; Reddy & Kumar, 2006). On the other hand, stakeholders on the shores are interested in keeping the lake level within a certain range to avoid floods and support recreational activities or environmental services. Increasing the lake storage to avoid irrigation deficits means increasing the risk of flooding and therefore some compromise needs to be established. Moreover, the optimal trade-offs might differ on a case-by-case basis, and they can be difficult to identify without a picture of all possible options.

Unfortunately, research work on optimizing a sequential decision problem often focuses on maximizing a single, scalar objective (Sutton & Barto, 2018). For the aforementioned real-world problems, this means somehow combining the different objectives into a single target metric. A possible option is for the decision maker to ask an expert to manually engineer a metric so to induce a specific behavior after optimization (Roijers et al., 2013). However, this approach has several drawbacks. First, such work can be expensive and error-prone, as it often requires domain expertise and extensive tuning before the optimization process finally produces a behavior that satisfies the decision maker. By focusing on a single solution at a time, this process also fails to inform the decision maker of all the trade-offs that could be possible. Finally, the decision maker has no way to convey directly their actual preferences, instead having to rely on the expertise of the designer to construct a suitable metric. These issues can severely restrict the influence that the decision maker has on the process, thus possibly missing their preferred solutions.

As such, we advocate the use of an explicitly multi-objective approach, through the use of Multi-objective reinforcement learning (MORL) (Hayes, Rădulescu, et al., 2021). Recalling our dam optimization problem, we can, by explicitly optimizing on electricity demand, irrigation deficits and risks of flooding, show the impact of the learned solution on these different objectives. Moreover, we can personalize the importance of each objective such that the different stakeholders reach a consensus. We note that the preferences of the decision maker, i.e., his utility, thus become part of the optimization process.

We outline a few motivating scenarios, demonstrating different manners of involvement of the decision maker.

1.2 Motivating scenarios for multi-objective approaches

It might be tempting to believe that we can view any multi-objective optimization problem as a single-objective one. After all, if we can design a single-objective reward signal that matches the preferences of the decision maker, we can use single-objective optimization to learn the optimal solution. However, Roijers et al. (2013) show that designing this function is not always feasible, possible or desirable. They outline 3 scenarios for which using multi-objective optimization is beneficial. Recently, Hayes, Rădulescu, et al. (2021) propose 3 additional scenarios on top of the ones from Roijers et al. (2013). We summarize these scenarios and classify them in 3 broad categories. Each category assumes different degrees of knowledge over the utility function during the search process. This is also reflected in the presented scenarios.

(a) In the *unknown utility function scenario*, it is not advisable to use a priori scalarization because the utility function is not known at the time of learning. In this situation, it is better to calculate a set of policies that cover different possible outcomes in order to be able to respond quickly if more information becomes available. For example, in wind farm management, it can be difficult to determine the best course of action because there are conflicting objectives - maximizing power output and minimizing maintenance costs caused by the strain of operation - and certain factors such as storms, the wake effect, and grid instability can impact the lifespan of turbine components (Verstraeten et al., 2019, 2020). Since the relationship between these factors and preventive control measures is not fully understood, it is important to learn a set of optimal solutions that can be quickly implemented when necessary.

(b) In the *decision support scenario*, the user's preferences are uncertain or difficult to determine, making it infeasible or impossible to use a priori scalarization because the user's utility function is unknown. In this situation, it is better to present the users with a set of policies from which they can choose based on their own preferences. For example, in water management, it can be challenging to identify the optimal way to manage a water reservoir because there are many stakeholders with different and potentially conflicting objectives. Each stakeholder has their own preferences for how the water should be managed, and these preferences can affect different aspects of businesses and communities located around the lake. It may be difficult or impossible to accurately capture the preferences of all stakeholders while considering the trade-offs across all objectives. Instead, it is more practical to learn a set of optimal policies and present them to the users for selection, possibly through a collective decision made by a local council or government.

The difference between this scenario and scenario (a) lies in the selection phase, after the solution set has been learned. In the first scenario, the utility function is explicitly defined a posteriori and applied on each solution of the solution set. In the second scenario, the users decide on their preferred solution without formalizing the utility function. In this case, the utility function is implicit. The second scenario

is typically applied for user selection since defining their utility explicitly is hard or not feasible.

(c) In the *known utility scenario*, the user's preferences are known in advance and quantifiable. We can thus work with the utility function and, if desired, apply scalarization and use single-objective optimization. However, this might not always be desirable as Roijers et al. (2013) have shown that using a priori scalarization can lead to an intractable problem. Moreover, using multiple reward signals provides additional information that can be used to speed up the learning process. We expand further upon this in this dissertation, see Chapter 3.

(d) In the *interactive decision support scenario*, we can interact with the decision maker during the learning process. There might be too much uncertainty about the utility function a priori. However, we can reduce this uncertainty during learning by eliciting the user's preferences. For example, by asking the user to rank some proposed solutions we can refine our knowledge over its utility, narrowing the search space. At the cost of requisitioning the user's time, we might be able to greatly reduce the amount of resources necessary to find the optimal solution.

(e) In the *dynamic utility function scenario*, the utility of the decision maker can change over time. It is thus undesirable to apply a priori scalarization. In this case, we would prefer to learn a solution set, such that the user can select an optimal alternative when its utility changes. An alternative would be to learn a single policy and dynamically adapt the utility function as it changes over time, but this can lead to instability during learning and a period of suboptimal behavior as the learner adapts, which need not occur if the learner has learned a set of solutions in advance. Depending on how often or how greatly the utility function changes, this might also be more costly in terms of computational resources (Abels et al., 2019; S. Wang et al., 2022).

(f) The *review and adjust scenario* can be seen as a generalization of the dynamic utility function scenario. In this scenario, not only is there uncertainty about the decision maker, there is also uncertainty about the MOMDP itself. Upon seeing the solution set, the decision maker might identify objectives that he previously missed, but now deems important.

All these scenarios show different use-cases where multi-objective optimization is advantageous. From an algorithmic perspective, many of the scenarios present similarities. For example, scenario (a), (b) and (e) all aim to learn a full coverage set. The difference lies in the selection phase. In a similar fashion, scenario (f) requires to learn a coverage set, although this might only be a partial set if the number of objectives increases. In all these scenarios, the utility function is unknown.

At the other side of the spectrum lies scenario (c), where we assume the utility function is known a priori. Since we only need to learn a single policy, and we can take advantage of our full knowledge over the utility function, we need another class of algorithms for this type of scenario.

Thus, from an algorithmic perspective, we observe two different approaches. Scenario (d) shows a third approach. Although the goal is to learn a single policy, we only have partial

knowledge over the user's utility. However, in this case we can refine this knowledge through interactions with the user. As such, we need algorithms that are able to propose solutions so we can improve this knowledge, instead of algorithms that learn the optimal solution with regard to the current estimate of the utility function.

In conclusion, we distinguish 3 general classes of multi-objective algorithms based on the amount of knowledge on the utility function:

1. Full-knowledge algorithms, for which we know the utility function. These are single-policy algorithms (Vamplew et al., 2011) that can directly optimize on the user utility. Nevertheless, keeping track of the multiple objectives separately can help in interpreting and understanding the policy, and increase sample-efficiency compared to using a single-objective algorithm, especially if the utility function exhibits complex, non-linear behavior.

2. Knowledge-gathering algorithms, for which interacting with the decision maker is key to improve this knowledge. These algorithms need to trade off the cost of involving the user and the cost of interacting with the environment.

3. Incomplete-knowledge algorithms, for which the utility function is unknown. These are called multi-policy algorithms (Vamplew et al., 2011), as they require to learn a solution for each possible trade-off.

1.3 Research question and contributions

In the previous section, we classified three general cases of multi-objective RL algorithms based on the amount of available knowledge regarding the utility function. This classification leads to the main research question of this dissertation. How can we best incorporate the available knowledge we have over the utility function in the multi-objective optimization process?

Each of the 3 classes of algorithms shown in Section 1.2 exhibit their own peculiarities and dilemmas. However, we can summarize them in one main challenge. Compared to single-objective optimization, where the objective to maximize is clear, multi-objective optimization is confronted with alternative choices. This greatly increases the complexity of the search for the optimal solution. Parts of the search-space that can quickly be discarded for single-objective problems cannot be discarded for their multi-objective counterpart, as they might hold interesting trade-offs. In general, the search towards the optimal solution is more exhaustive, as it is unclear what the exact preferences of the decision maker are.

Thus, the uncertainty over the decision maker's preferences are the principal cause of this exhaustive search. Hence, it seems logical that incorporating all available prior knowledge over these preferences, even if incomplete, will narrow down the search space. Already, most prior and current work in MORL makes assumptions on how these preferences are shaped, estimating that weighting each objective by a percentage of importance is enough to model the decision maker's utility (Abels et al., 2019; Alegre et al., 2023; Alegre et al., 2022; Barrett & Narayanan, 2008; Hiraoka et al., 2009; Mossalam et al., 2016; Roijers et al., 2015; R. Yang et al., 2019). However, this is mostly due to the compatibility of weighted objectives with the RL framework, not because it is representative of user preferences (Vam-

plew et al., 2022). Indeed, most common types of utility functions have different shapes, with different properties (Arrow et al., 1961; Carbaugh, 2013; Douglas, 1976).

In this dissertation, we take a utility-centered approach, first incorporating prior knowledge over the utility function, and then designing algorithms that best use this knowledge to efficiently learn the optimal solution with respect to the decision maker's preferences. We propose contributions for different kinds of prior knowledge, ranging from full knowledge over the utility function to no prior knowledge at all.

1.3.1 Known, non-linear utility functions

Although, as stated previously, a large part of the MORL literature assumes we can express the user utility as a weighted sum over objectives, this does not align with human preferences which often are non-linear. In this case, even if the utility function is known a priori, dedicated multi-objective models and methods are required (Roijers et al., 2013). We propose *Multi-Objective Categorical Actor-Critic* (MOCAC), an algorithm for known utility functions, which corresponds to the 1st general class of multi-objective algorithms. MOCAC is a method that learns the optimal solution, even when the utility function is non-linear, by learning a multi-variate distribution over the multi-objective returns, inspired by distributional reinforcement learning (Bellemare et al., 2017). Moreover, by leveraging the additional information provided by the different objectives, it does so more efficiently than single-objective approaches that directly optimize the utility.

1.3.2 Partially known utility function, with interactive learning

In contrast with the 1st general class of multi-objective algorithms, often it is not possible to have full knowledge over the utility function, and we only have partial prior knowledge. However, we might be able to interact with the decision maker to improve our partial understanding of his or her preferences. We propose *Multi-Objective Partially Observable Monte-Carlo Planning* (MOPOMCP), an algorithm that optimizes the timing of user-interaction, such that it maximally improves our chances of learning the best solution, within the available (computational, interaction) resources. We do this using a Bayesian approach (Russo, 2016), keeping a belief distribution over the possible utility functions given our current knowledge, and timing the interactions such that they best improve our belief.

1.3.3 Unknown, non-linear utility function

In the case that we do not have any prior knowledge over the utility function, and we cannot interact with the decision maker to improve our understanding over its utility, we need to learn the set of all possible trade-offs, guaranteeing that the learned set contains the decision maker's preferred solution. We propose *Pareto Conditioned Networks* (PCN), an algorithm that efficiently learns all optimal trade-offs, by feeding the experience used for different compromises into a single neural network, thus allowing to share experience across policies (Kumar et al., 2019; Schmidhuber, 2019).

Although MORL algorithms for unknown utility functions have been investigated, they often make assumptions on the types of preferences the decision maker can have (see

Section 2.4) (Abels et al., 2019; Alegre et al., 2023; R. Yang et al., 2019). In contrast, our method makes no assumption on the shape of the utility function. Moreover, existing MORL algorithms with no assumption on the shape of the utility function scale poorly with the complexity of the problem (Moffaert & Nowé, 2014; Parisi et al., 2016), while our approach is scalable with the size of the decision problem. Finally, our approach is also scalable with respect to the number of objectives.

1.3.4 Learning mitigation policies for epidemic outbreaks

As shown by the COVID-19 pandemic, infectious disease outbreaks represent a major global challenge (P. J. K. Libin et al., 2021). We present a real-world use-case for multi-objective reinforcement learning, by learning multi-objective mitigation strategies for epidemic outbreaks. We focus on policies putting restrictions on social contacts, to reduce the spread of the epidemic and the consequent burden on hospitals. We make no assumptions on the utility function, learning all trade-offs between hospitalizations and loss in social contacts, by extending our previous contribution, PCN, to this setting. We show that our methodology can be used to learn a wide set of high-quality policies on real-world problems, providing the decision maker with insightful and diverse alternatives.

1.4 Organization of this dissertation

This dissertation has been organized in logical, rather than chronological order. It follows the outlined contributions, ordered in decreasing amount of knowledge over the utility function.

Chapter 2 presents a brief introduction on multi-objective reinforcement learning and its solution concepts. Chapter 3 presents our contributions for known, non-linear utility functions, which has been published in (Reymond, Hayes, et al., 2023). Chapter 4 presents recent contributions for the interactive setting with the decision maker, which has not yet been published. In Chapter 5, we present our novel algorithm for unknown utility functions, which has been published in (Reymond, Eugenio, & Nowè, 2022). Chapter 6 showcases a real-world application of this algorithm for epidemic outbreaks, available as a preprint at (Reymond, Hayes, et al., 2022), and currently under review at Expert Systems with Applications[1]. Finally, we conclude our dissertation in Chapter 7 and outline directions for future work.

A complete list of publications is available in Section 7.1.3. Next to the publications incorporated in this manuscript, Reymond et al. (2018) includes initial work with multiple objectives and preliminary work in workshops (Reymond & Nowe, 2019; Reymond et al., 2021, 2023). Moreover, our research resulted in collaborative work, which is cited in the related work sections of this manuscript (Hayes et al., 2021, 2023; Roijers et al., 2021; Wang et al., 2022). One of these collaborations led to an overview paper co-authored by numerous researchers of the MORL community (Hayes et al., 2022). Finally, this list contains published work not related to MORL, and as such are not cited in this document (Avalos et al., 2022a, 2022b; Nevens et al., 2018).

[1]https://www.sciencedirect.com/journal/expert-systems-with-applications

Chapter 2

Background

2.1 Multi-objective reinforcement learning 13
2.2 Multi-objective Markov decision processes 14
2.3 Policies 14
2.4 Utility function 15
2.5 Optimality criterion 17
2.5.1 The special case of linear scalarization 18
2.6 Coverage sets 19
2.6.1 Linear utility functions under SER and ESR 20
2.6.2 Monotonically increasing utility functions under SER 21
2.6.3 The impact of stochastic policies on the coverage set 21
2.7 The utility-based perspective 23
2.8 Performance metrics for multi-objective optimization 24
2.8.1 Hypervolume 24
2.8.2 Expected Utility Metric 25
2.8.3 Expected Utility Loss 25
2.8.4 Maximal Utility Loss 25
2.8.5 The ε-indicator 26
2.8.6 The $\varepsilon - mean$-indicator 26

2.1 Multi-objective reinforcement learning

This chapter lays the fundamental background concepts necessary to understand this dissertation.

In this chapter, we formalize the MORL paradigm, introducing the framework used to formalize the sequential decision problem, the optimization criterions and the role of the decision maker in the learning process.

2.2 Multi-objective Markov decision processes

Recalling the water reservoir optimization example, we call in RL terminology the controller that regulates the water flow an agent. This agent interacts with an environment, in this case the water reservoir. These interactions happen through actions the agent can perform, e.g. opening or closing the dam gates. In order to decide which action to perform, the agent can observe the current state of the environment, e.g. the water level of the reservoir. Performing actions changes the state, as such the agent is able to observe the consequences of its actions. Finally, in order to be able to judge the quality of its actions, the agent receives a reward in the form of a numerical signal. For example, it could receive a reward of 1 if the demand is met, and -1 if it is not met. In real-world problems, there can often be multiple reward signals, e.g. one for electricity demand, one for agricultural needs, and one for environmental services.

Formally, we model these sequential decision problems as a Multi-objective Markov decision process (MODMP). A MOMDP is a tuple, $\mathfrak{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \boldsymbol{\mathcal{R}}, n \rangle$, where $\mathcal{S}, \mathcal{A}$ are the state and action spaces respectively, $\mathcal{T} \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a probabilistic transition function, γ is a discount factor determining the importance of future rewards and $\boldsymbol{\mathcal{R}} \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^n$ is an n-dimensional vector-valued immediate reward function, with n being the number of objectives of the problem. In single-objective RL, $n = 1$ while in *multi-objective reinforcement learning (MORL)*, $n > 1$.

2.3 Policies

In MORL, problems are thus defined as MOMDPs. While MOMDPs define the actions an agent can take, it does not define which actions are taken in each state. We call this the agent's *policy*. This policy dictates the agent's behavior. More formally, a policy π is a probabilistic function $\mathcal{S} \times \mathcal{A} \to [0, 1]$ determining, for each state $s \in \mathcal{S}$, the probability of taking an action $a \in \mathcal{A}$.

When the number of objectives $n = 1$, the goal is to find the optimal policy π^* that maximizes the expected sum of discounted rewards:

$$\pi^* = \arg\max_{\pi} \mathbb{E} \left[\sum_{t=0}^{h} \gamma^t r_t | \, \pi, s_0 \right], \tag{2.1}$$

where t is the timestep and s_0 the initial state of the environment, i.e. the start-state.

The expected sum of discounted rewards is often also called the Value, and reflects the quality of a policy:

$$V^{\pi} = \mathbb{E} \left[\sum_{t=0}^{h} \gamma^t r_t | \, \pi, s_0 \right]. \tag{2.2}$$

A policy π_1 is considered better than another policy π_2 when its Value is higher:

$$\pi_1 > \pi_2 \iff V^{\pi_1} > V^{\pi_2}. \tag{2.3}$$

In the multi-objective case, when $n > 1$, the Value is an n-dimensional vector, which we denote $\boldsymbol{V}$.

While this difference might seem relatively simple — we consider multiple criteria instead of one but the problem itself remains unchanged — it can completely transform the way to tackle the problem at hand.

To its core, it changes one of the most basic concepts of optimization. The goal of single-objective RL is to find the policy that maximizes its objective, the expected sum of discounted rewards. But with the introduction of multiple objectives, this notion of maximization (the $\arg\max$ term in Equation 2.1) becomes unclear.

Say we have a 2-objective MOMDP where finding a solution that improves one objective comes at the cost of the other objective. It becomes impossible to improve all the objectives at once. Which solution is then considered the optimal one? Without any additional information, there is no clear winner (e.g., we cannot decide which solution is optimal between $(0, 10)$, $(3, 3)$ or $(10, 0)$). This means that in multi-objective optimization there is no perfect solution, we need to make trade-offs between the different criterions.

However, to the end-user trying to solve a problem, some of these trade-offs are better than others. In this regard, an ordering between multi-objective solutions can be made. This ordering depends on the personal preferences of the end-user. In this manner, we can tie our multi-objective problem back to single-objective optimization: we aim to find the solution that is best according to the personal preferences of the end user.

2.4 Utility function

Although a MOMDP possesses multiple objectives, we can rank the compromises according to the end-user's preferences, or *utility*. We thus assume that there exists, either implicitly or explicitly, the existence of a scalarization function that converts a vectorial return to a scalar preference score. We call this function the *utility function* $u : \mathbb{R}^n \to \mathbb{R}$.

As a minimal assumption we consider that a solution that improves one of the objectives without hampering any of the other objectives will have a higher utility for the decision maker. We thus assume that the utility function is *monotonically increasing*:

$$u(\boldsymbol{V}^1) > u(\boldsymbol{V}^2) \iff (\forall i : V_i^1 \geq V_i^2) \wedge (\exists i : V_i^1 > V_i^2). \tag{2.4}$$

Often, diverse types of parametrized scalarization functions are used in MORL to represent the utility function, where the parameters represent the preferences of the decision maker.

The most commonly used type of scalarization function is the *linear scalarization* function (Abels et al., 2019; Alegre et al., 2023; Alegre et al., 2022; Barrett & Narayanan, 2008; Hiraoka et al., 2009; Mossalam et al., 2016; Roijers et al., 2015; R. Yang et al., 2019). This function makes a weighted sum over the objectives, where the weights are the parameters of the function. Each weight represents a percentage of importance for each objective.

Thus, the weight-vector is constrained to be positive and sum to 1, i.e. it is drawn from the $n-1$ dimensional simplex:

$$u(\boldsymbol{V}) = \boldsymbol{w}^\top \boldsymbol{V}, \{\boldsymbol{w} \in \mathbb{S}^{n-1}\}, \tag{2.5}$$

with $\mathbb{S}^{n-1}$ the $n-1$ dimensional simplex:

$$\mathbb{S}^{n-1} = \{\boldsymbol{w} \in \mathbb{R}^n : w_0 + \cdots + w_{n-1} = 1, w_i \geq 0 \text{ for } i = [n]\}. \tag{2.6}$$

Next to the ease of implementation, its main advantage lies in its additivity property. We can apply the utility function to the individual vectorial rewards, and its sum will be equal to the utility over the whole return:

$$\begin{aligned} u(\boldsymbol{R}) &= \boldsymbol{w}^\top \sum_{t=0}^{T} \boldsymbol{r}_t \\ &= \sum_{t=0}^{T} \boldsymbol{w}^\top \boldsymbol{r}_t \\ &= \sum_{t=0}^{T} u(\boldsymbol{r}_t). \end{aligned} \tag{2.7}$$

This means that without any additional changes, we can use single-objective RL algorithms to optimize directly on the utility, while keeping the convergence guarantees and properties of the original algorithm.

Although this additivity property is a considerable advantage, linear scalarization comes with some issues. One of its main disadvantages is that, due to linear functions being a subset of all monotonically increasing functions, it cannot represent all possible preferences of the decision maker. Consider our previous example with 3 possible trade-offs, $\boldsymbol{V}^1 = (10, 0)$, $\boldsymbol{V}^2 = (3, 3)$ and $\boldsymbol{V}^3 = (0, 10)$, no weight-parameters $\boldsymbol{w}$ exist such that $\boldsymbol{V}^2$ becomes the preferred solution. The weights $\boldsymbol{w} = (\frac{1}{2}, \frac{1}{2})$ that consider both objectives as equally important have utilities $u(\boldsymbol{V}^1) = 5$, $u(\boldsymbol{V}^2) = 3$ and $u(\boldsymbol{V}^3) = 5$. We expand more on this limitation in Section 2.6.

To alleviate this issue, another type of parametrized scalarization function can be used, the *weighted-norm scalarization*. It computes the weighted l_p norm of a solution, where the parameter weights $\boldsymbol{w}$ are constrained in the same fashion as for linear scalarization:

$$\begin{aligned} u(\boldsymbol{V}) &= (\sum_{i}^{n} w_i |V_i|^p)^{\frac{1}{p}} \\ &= ||\boldsymbol{w}^{\frac{1}{p}} \boldsymbol{V}||_p, \{\boldsymbol{w} \in \mathbb{S}^{n-1}\}. \end{aligned} \tag{2.8}$$

A special case of this scalarization function which is often used in the literature is the *Chebyshev scalarization* (Van Moffaert et al., 2013a). Chebyshev scalarization computes the l_∞ norm between a point in the multi-objective space and a utopian point $\boldsymbol{z}^*$:

$$u(\boldsymbol{V}) = \max_i w_i |V_i - z_i^*|, \{\boldsymbol{w} \in \mathbb{S}^{n-1}\}. \tag{2.9}$$

While the Chebyshev scalarization allows to recover all possible trade-offs, it requires to choose an utopian point.

Although it might seem convenient to use weights to define the importance of each objective, in practice the relationship between weights and policy may be unpredictable, as a small change in weights might result in completely different policies (Vamplew et al., 2011). Moreover, humans find it challenging to express their preferences in absolute terms (e.g., "70% importance for objective 1"), as using numbers to express preferences can be unnatural and prone to errors (Tesauro, 1988).

2.5 Optimality criterion

In single-objective RL, we have defined the policy with the highest expected return as the optimal policy π^*. Similarly, since we assume the existence of a utility function u in MORL, we also assume the existence of a single optimal policy. However, depending on where we apply u, we can define 2 different optimization criteria for π^* (Hayes, Rădulescu, et al., 2021). As these criteria can result in vastly different optimal policies, one should choose the most appropriate one depending on the problem at hand.

As a first criterion the agent aims to compute a policy π that optimizes the utility of the expected vectorial return, i.e.,

$$\pi^* = \arg\max_\pi u \left(\mathbb{E} \left[\sum_{t=0}^{h} \gamma^t \boldsymbol{r}_t | \pi, s_0 \right] \right). \tag{2.10}$$

This is known as the *scalarized expected return* (SER) optimization criterion and is the one used in most MORL research (Roijers et al., 2013). The particularity of this criterion is that the utility is optimal on the average of multiple executions of the learned policy. One advantage is that, since u is applied on the vectorial Value, optimizing under SER allows to apply temporal-difference on each objective separately and take advantage of classic value-based methods (Van Moffaert et al., 2013b).

As a second criterion the agent aims to maximize the expected utility over single policy executions, i.e.,

$$\pi^* = \arg\max_\pi \mathbb{E} \left[u \left(\sum_{t=0}^{h} \gamma^t \boldsymbol{r}_t \right) | \pi, s_0 \right]. \tag{2.11}$$

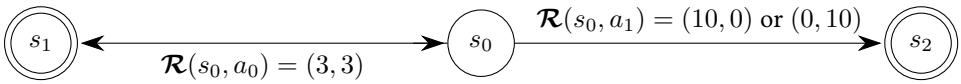


Figure 2.1: A MOMDP for which the optimal policy differs depending on the optimization criterion with the utility function $u = \min(r_0, r_1)$. Under ESR, $\pi^*(a_0 \mid s_0) = 1$ while under SER, $\pi^*(a_0 \mid s_1) = 1$.

This is the expected scalarized return (ESR) criterion. Under ESR u is applied directly on the episodic return. Since u can be non-linear, different approaches are required for ESR than for SER.

We illustrate the difference between those two solution concepts using a small example, shown in Figure 2.1. Consider a single-step, 2-objective MOMDP with two possible actions a_0, a_1. Action a_0 always results in reward $\boldsymbol{r} = (3, 3)$, while action a_1 results with equal probability in either $\boldsymbol{r} = (0, 10)$ or $\boldsymbol{r} = (10, 0)$. Under ESR, given $u = \min(r_0, r_1)$, the optimal policy is to always take a_0 since it results in the highest utility $u = 3$ (compared to $u = 0$ for a_1). However, *in expectation*, i.e., over many policy runs, $\mathbb{E}[\boldsymbol{r}|a_1] = (5, 5)$. Under the SER criterion, given the same utility function, the optimal policy is to always take a_1.

For example, say we are optimizing the daily commute to work of the decision maker. The resulting policy should be optimized under SER if he has a flexible start-time allowing him to arrive later some days, as long as he arrives early other days to compensate. However, if his employer requires him to be precisely on time every day, ESR should be used since arriving on time depends on every policy execution.

2.5.1 The special case of linear scalarization

Since the optimal policy can vary greatly depending on the optimization criterion used, it is crucial to decide which criterion to optimize on in order to learn the optimal policy that accurately reflects the preferences of the decision maker. However, one class of utility functions exists for which this choice does not matter as both optimization criterions lead to equivalent optimal policies. This is the case of linear scalarization functions (Equation 2.5). Due to their linear additivity property (Equation 2.7), both ESR and SER become equivalent (Rădulescu et al., 2020):

$$\begin{aligned}
\mathbb{E}\left[u\left(\sum_{t=0}^{h}\gamma^t \boldsymbol{r}_t\right)|\pi, s_0\right] &= \mathbb{E}\left[\boldsymbol{w}^\top \sum_{t=0}^{h}\gamma^t \boldsymbol{r}_t|\pi, s_0\right] \\
&= \boldsymbol{w}^\top \mathbb{E}\left[\sum_{t=0}^{h}\gamma^t \boldsymbol{r}_t|\pi, s_0\right] \\
&= u\left(\mathbb{E}\left[\sum_{t=0}^{h}\gamma^t \boldsymbol{r}_t|\pi, s_0\right]\right).
\end{aligned} \tag{2.12}$$

Not only does the linear additivity property allow to apply the utility function on individual rewards, using linear scalarization allows optimizing on both SER and ESR equivalently. These properties are the reasons why linear scalarization is a popular choice of scalarization functions in MORL.

2.6 Coverage sets

Ultimately, the goal in MORL is to learn a policy π that best fits the preferences of the decision maker. We do so through the use of the utility function u and the SER or ESR optimality criterion. Both these criterions assume full knowledge over u in order to learn π. Although we have mentioned common scalarization functions, we have until now avoided the question of how to obtain this knowledge over u. This is quite a hard question, as user preferences come with multiple unknowns.

First, preferences are often shaped by a multitude of factors, such as personal experiences, upbringing, individual tastes, and values. These factors can vary significantly from person to person, which makes it challenging to create a standardized approach to understanding and measuring preferences. This makes user preferences complex.

Furthermore, preferences can also be influenced by context and situational factors. For example, a person's preference for a particular type of food may vary depending on the time of day, the weather, or their mood. Similarly, a person's preference for a particular type of entertainment may vary depending on their current interests or circumstances. Thus, user preferences are difficultly quantifiable.

Finally, preferences can evolve and change over time as a person's experiences, values, and priorities shift. What was once a favorite food or activity may no longer hold the same appeal, or a person may develop a new appreciation for something they previously disliked.

Thus, it is not straightforward to transform the multi-objective problem back to a single-objective one, and it depends in great part on our knowledge over these preferences. Indeed, possessing full knowledge over the utility function allows to directly optimize on the SER or ESR criterion. In this way, we can focus our search to only the trade-offs relevant to the end-user and learn a single, personalized policy. On the contrary, if we have no knowledge over the utility function, then any compromise could be the optimal solution. To guarantee that we can provide the optimal policy regardless of the preferences of the decision maker we need to learn a set that, for each possible utility function, contains the corresponding optimal policy. Such a set "covers" all possible utility functions, as such we call it a *coverage set*.

Coverage sets are an important concept in MORL as often we assume that the preferences of the decision maker are unknown, and aim to learn the whole coverage set. In this setting, we thus try to learn multiple policies, in contrast to the learning of a single optimal policy in single-objective optimization.

Concretely, we say that a policy π_1 *dominates* another policy π_1 if its utility is higher, for all possible utility functions $u \in \mathcal{U}$:

$$\pi_1 \succ \pi_2 \iff (\forall u \in \mathcal{U} : V_u^{\pi^1} > V_u^{\pi^2}). \tag{2.13}$$

We note that the scalarized V_u depends on the optimization criterion (it should be $V_{u,\text{ESR}}$ or $V_{u,\text{SER}}$) but the ESR, SER suffix has been dropped, as the dominance concept is valid for both optimization criterions.

This allows us to define the set of all non-dominated policies, which contains all the policies that are not dominated by any other policy, i.e., there exists a utility function for each of the policies in this set, for which that policy is optimal:

$$\Pi^* = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : \boldsymbol{V}^{\pi'} \succ \boldsymbol{V}^{\pi}\}. \tag{2.14}$$

In general, we call any set of policies a *solution set*, and any solution set that only contains non-dominated policies a *coverage set*. The coverage set that contains all possible non-dominated policies is called the *optimal coverage set* Π^*.

It is important to note here that this concept of optimal coverage set depends on the type of optimality criterion (SER, ESR) to use and the class of utility functions that represent the decision maker (from monotonically increasing utility functions to linear utility functions). Thus, we can further refine the definition of coverage set and learn sets of policies specific to the problem setting at hand.

By combining the 2 possible optimality criterions and the 2 covered classes of utility functions we obtain 4 combinations of possible types of coverage sets. However, we have seen in Section 2.5.1 that for linear utility functions SER and ESR become equivalent criterions, thus we can further reduce the number of combinations to 3. The solution concept for the coverage set of non-linear utility functions under ESR has only been recently defined and is not used in this dissertation, so we refer the interested reader to Hayes, Verstraeten, et al. (2021) for further explanations.

2.6.1 Linear utility functions under SER and ESR

Using a weighted sum as utility function restricts the number of non-dominated solutions that can be learned. When using linear weights, a solution that is on the concave part of the coverage set always has a lower utility than a neighboring solution on the convex part, regardless of the chosen weights (Roijers et al., 2013). Thus, we can only discover the solutions that are on the convex part of the coverage set.

$$C = \{\pi \in \Pi \mid \exists \boldsymbol{w} \in \mathbb{S}^{n-1} : \boldsymbol{w}^\top \boldsymbol{V}^{\pi} \geq \boldsymbol{w}^\top \boldsymbol{V}^{\pi'}, \forall \pi' \in \Pi\}. \tag{2.15}$$

Thus, if we know that the optimal coverage set of our problem is convex, using linear scalarization is appropriate as by definition it optimizes for the convex coverage set. This holds even if the ground-truth utility function of the decision maker is non-linear, as its optimal solution is still contained in the convex coverage set.

2.6.2 Monotonically increasing utility functions under SER

Consider a problem where we have no information concerning the utility function of the decision maker and the optimal coverage set can be concave or partially concave. Then, to ensure that the optimal policy is contained in the optimal solution set, we should optimize for monotonically increasing utility functions. Under SER, the utility function is applied on the vectorial Value $\boldsymbol{V}$. Without any additional information about the utility function, we can only guarantee that a policy is better than another if its Value is higher *across all objectives* than the Value of the other policy. A solution for which it is impossible to improve one of the objectives without hampering another is said to be *Pareto-efficient*.

More formally, a policy π_1 is said to *Pareto-dominate* another policy π_2 if its expected return $\boldsymbol{V}^{\pi_1}$ is higher or equal across all objectives than $\boldsymbol{V}^{\pi_2}$, and there exist at least an objective where $\boldsymbol{V}^{\pi_1}$ is better than $\boldsymbol{V}^{\pi_2}$:

$$\pi_1 \succ_P \pi_2 \iff (\forall i \in [n] : V_i^{\pi_1} \geq V_i^{\pi_2}) \wedge (\exists i : V_i^{\pi_1} > V_i^{\pi_2}), \tag{2.16}$$

where $\succ_P$ is the *Pareto-dominance* operator. Notice that this definition does not use the utility function u since u is unknown. Thus, in this setting, we directly focus on the Pareto-efficient vectors. We call the corresponding coverage set the *Pareto front*:

$$P = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : \pi' \succ_P \pi\}. \tag{2.17}$$

The advantage of using this coverage set lies in its genericity, as it is guaranteed to contain the optimal policy for any kind of monotonically increasing utility function. On the downside, this set can be prohibitively large, in which case we may no longer be able to compute all its corresponding policies without soliciting more information on how to prioritize the objectives. Gaining more insight allows us to scale down the search, with at the other end of the spectrum the class of linear utility functions and its corresponding convex coverage set.

Taking full advantage of our knowledge over the user utility function can thus greatly reduce the search space. Thus, as is advocated by Roijers et al. (2013), we argue that the utility function should be a central part of the multi-objective optimization process and adopt a *utility-based perspective*.

2.6.3 The impact of stochastic policies on the coverage set

Equation 2.1 shows that, for single-objective optimization, the optimal policy is the policy leading to the highest expected return. Since this policy greedily takes the action leading to the highest V-value, it is deterministic.

In contrast, consider a set of 2 optimal trade-offs, $\boldsymbol{V}^1 = (10, 0)$, $\boldsymbol{V}^2 = (0, 10)$, and their associated deterministic policies π^1, π^2, respectively. We can create a new, stochastic policy π^3 that, in the start-state s_0, randomly chooses to follow either π^1 or π^2 for the duration of the episode, with probability $p_{\pi^1}, 1-p_{\pi^1}$, respectively. Then $\boldsymbol{V}^3 = p_{\pi^1}\boldsymbol{V}^1 + (1-p_{\pi^1})\boldsymbol{V}^2$. For example, using $p_{\pi^1} = 0.5$ results in a new, non-dominated solution $\boldsymbol{V} = (5, 5)$. By varying p_{π^1}, we obtain infinitely many possible trade-offs, showing that, in multi-objective optimization, stochastic policies can lead to new, non-dominated solutions.

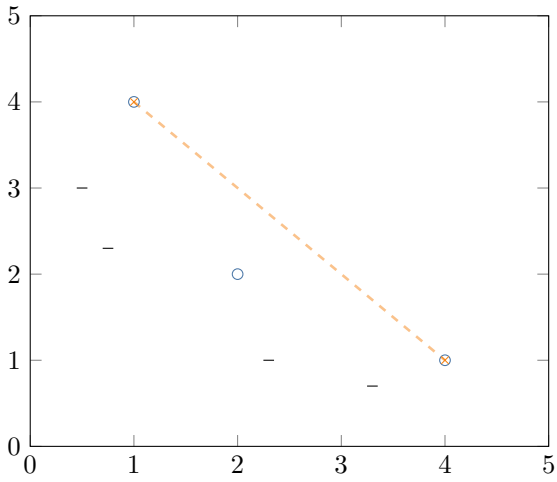


Figure 2.2: Distinction of different coverage sets. For any setting, the points marked with "-" are suboptimal, as they are dominated by the blue points with circle markers. When the utility function is non-linear, the optimal coverage set corresponds to the Pareto front, in blue. However, for linear utility functions, the point at position $(2, 2)$ is suboptimal, as no weight $\boldsymbol{w} \in \mathbb{S}^{n-1}$ exist that results in this point having a higher utility than any of the orange points (marked with "x"). Moreover, even for non-linear utility functions, if stochastic policies are allowed, then combining the policies leading to the points marked with "x" results in Values on the orange dashed line, which also dominates the point at position $(2, 2)$.

Moreover, we observe that, by using these stochastic policies, solutions on the concave part of the coverage set (e.g., $(3, 3)$) become dominated by the stochastic mixture of solutions on the convex part. In this case, as for linear utility functions, the convex coverage set (Equation 2.15) is the optimal solution set (Vamplew et al., 2009).

Thus, the optimal solution set when allowing stochastic policies is the convex coverage set. Nonetheless, for many settings, it is undesirable to have stochastic policies. For example, in a medical treatment planning setting, the patients would probably object to random selection of different medicines. Or, in the management of a hydroelectric power plant, the decision maker does not want to be presented with a policy that has a probability of completely draining the water reservoir even if that policy is optimal, as it would have catastrophic consequences for nearby towns (Hayes, Rădulescu, et al., 2021).

As such the type of policy (deterministic or stochastic) should be considered on a case-by-case basis, depending on the problem setting. An illustration of the different types of coverage sets is shown in Figure 2.2.

2.7 The utility-based perspective

The end-goal of a multi-objective optimization problem is to maximize the user utility. Since the properties of a user's utility may drastically alter the desired solution, taking advantage of our knowledge over this utility is of paramount importance.

The utility-based approach is in contrast to the earlier axiomatic-based approach (Moffaert & Nowé, 2014; White, 1982). In the axiomatic-based approach, the goal is to learn the Pareto front of the decision problem. Since the Pareto front is the most generic coverage set, this ensures that, with the axiomatic-based approach, we learn the optimal solution. However, this set can be too large, making it prohibitively expensive to retrieve. Instead, we should design algorithms that encompass knowledge over the utility function or the relevant type of coverage set. This is a viable strategy for many practical applications, where information about the user's preferences is available through domain knowledge. The axiomatic approach would disregard this knowledge leading to unnecessary use of resources.

Thus, the utility-based approach consists in first gathering all possible knowledge over the user's preferences and over the problem at hand to narrow the search process. Then, a suitable algorithm is selected to discover the appropriate solution or solution set. Finally, we choose an appropriate method to extract the preferred solution from the solution set. Concretely:

1. We collect all possible a priori knowledge over the user's utility. This information will help us determine the class of utility functions we will employ. Throughout this dissertation, we will assume different amounts of a priori knowledge, and see that this results in different approaches.

   One key element of prior knowledge we require, is whether we should optimize on the ESR or SER optimization criterion described in Section 2.5.

2. We decide whether deterministic or stochastic policies are allowed for the problem at hand. Depending on the problem, we might not desire stochastic policies, however, if possible, we should use them, as stochastic policies can be strictly better than deterministic policies (Vamplew et al., 2009; White, 1982).

3. Based on the previous points, we decide on the optimal solution concept. For examples, using stochastic policies allows us to optimize the convex coverage set (Section 2.6.3).

4. We select or design a MORL algorithm that fits this solution concept. Due to the incorporation of prior knowledge, different settings arise. Taking advantage of the assumptions made for each setting result in specialized algorithms. While this means we need to design MORL algorithms for each setting, these algorithms result in more efficient learning than a generic approach.

5. When the utility function is unknown and the solution set is composed of multiple policies, we devise a method to extract the optimal policy from the solution set. One way would be to ask the decision maker to choose its preferred trade-off out of all the ones present in the solution set. However, this might not be possible when the

number of alternatives is too large. In this case, it might be better to estimate the decision maker's utility, e.g., through preference elicitation (Zintgraf et al., 2018).

2.8 Performance metrics for multi-objective optimization

We have seen in the taxonomy that there exist quite a few scenarios for multi-objective optimization, resulting in a wide range of possible algorithms. However, until now we have eluded how we can assess the performance of these algorithms and how we can compare them with one another.

In single-objective RL, the optimal policy π^* is the one with the highest possible Value V so V is typically used as a performance metric. Analogously, when the utility function u is known, we can use the ESR and SER optimality criterions as performance metrics. In contrast, we have seen that when u is unknown our goal is to learn coverage sets that contain the optimal policy for every possible utility function.

Comparing the learned coverage sets of different algorithms is a non-trivial task, as one algorithm's output might dominate the other in some part of the objective-space, but be dominated in another. Intuitively, one would generally prefer the algorithm that obtains better returns for a wider range of utility functions.

2.8.1 Hypervolume

The most widely used metric in the literature is called the *hypervolume* (Mannion et al., 2018; Vamplew et al., 2011; W. Wang & Sebag, 2013; Yliniemi & Tumer, 2016; Zitzler & Thiele, 1999). This metric evaluates the learned coverage set Π by computing its volume w.r.t. a fixed specified reference point $\boldsymbol{p}$. It can be seen as the union of boxes delimited by the reference point and the solutions in Π:

$$H(\Pi) = \Lambda\left(\bigcup_{\pi\in\Pi}[\boldsymbol{p}, \boldsymbol{V}^\pi]\right) \tag{2.18}$$

where Λ is the Lebesgue measure (also called n-dimensional volume) and $[\boldsymbol{p}, \boldsymbol{V}^\pi] = \{\boldsymbol{q} \in \mathbb{R}^n \mid \boldsymbol{q} \geq \boldsymbol{p} \wedge \boldsymbol{q} \leq \boldsymbol{V}^\pi\}$ is the box delimited below by the reference point $\boldsymbol{p}$ and above by $\boldsymbol{V}^\pi$.

The reference point is taken as a lower bound on the achievable returns so that the volumes are always positive. Thus, the hypervolume envelops all possible V-values that are dominated by that coverage set, with more dominating coverage sets having a larger hypervolume. This metric is by definition the highest for the Pareto front, as no other possible solution can increase its volume (since they are all dominated). While the hypervolume metric is widely used and does give a measure of the coverage of a solution set, it can be difficult to interpret. The benefit of a certain increase or decrease in hypervolume is not readily apparent to the end user, and does not necessarily correlate to significant changes in expected utility. When working in high-dimensional objective-spaces, adding or removing a single point can lead to wildly different hypervolume values, especially if the point

lies close to an extremum of the space. Moreover, the hypervolume assumes that every Pareto-non-dominated point can contribute to the user utility. This is not necessarily the case. For example, we have seen in Section 2.6.1 that the optimal coverage set for linear scalarization is the convex coverage set, for which all concave solutions do not contribute to the user utility.

2.8.2 Expected Utility Metric

The hypervolume does not necessarily correlate with the utility of the user. To alleviate this, we can use the *Expected Utility Metric* (EUM) (Zintgraf et al., 2015). As the name implies, it measures the expected utility for a user given a solution set:

$$EUM(\Pi) = \mathop{\mathbb{E}}_{u \sim \mathcal{P}(\cdot|\mathcal{U})} \left[\max_{\pi \in \Pi} u(\boldsymbol{V}^{\pi}) \right], \tag{2.19}$$

where $\mathcal{P}$ is the probability distribution over the utility functions. EUM explicitly measures the performance of a coverage set Π across all possible utility functions $u \in \mathcal{U}$. Contrary to the hypervolume, an improvement in EUM leads to a similar improvement in utility. One downside of this metric is that we explicitly need the set of all possible utility functions $\mathcal{U}$, which is not always available or easy to obtain. Nonetheless, it can be used with any of the classes of utility functions defined in Section 2.4, such as linear scalarization (Equation 2.5) or Chebyshev scalarization (Equation 2.9). Note however, that for these classes of utility functions, the probability distribution $\mathcal{P}$ might not be uniform over the weight-space. This is another downside of using EUM, it requires accurate knowledge over $\mathcal{P}$.

2.8.3 Expected Utility Loss

The Expected Utility Metric provides a way to measure how much utility we can expect from a given solution set. But we do not necessarily know when a certain value is satisfactory for the decision maker. This makes the EUM sometimes hard to interpret. As an alternative, we can measure how far off we are from the best possible coverage set. This can be meaningful since it provides a relative measure with respect to the optimal solution. We call this variant the *Expected Utility Loss* (EUL):

$$EUL(\Pi) = \mathop{\mathbb{E}}_{u \sim \mathcal{P}(\cdot|\mathcal{U})} \left[\max_{\pi^* \in \Pi^*} u(\boldsymbol{V}^{\pi^*}) - \max_{\pi \in \Pi} u(\boldsymbol{V}^{\pi}) \right]. \tag{2.20}$$

As a downside, EUL requires to know the true optimal coverage set Π^*, so in theory it can only be used in test problems with a known structure. In practice, we can construct an approximation of the optimal coverage set for a specific problem by selecting the non-dominated policies across all our experiments. However, in this case EUL can only be computed as an evaluation metric a posteriori, and cannot be used during the learning process to guide the search of an algorithm towards better coverage sets.

2.8.4 Maximal Utility Loss

An alternative utility-centered metric that does not require knowledge over $\mathcal{P}$ is the *Maximal Utility Loss* (MUL) (Zintgraf et al., 2015). It measures the maximal loss in utility that

occurs when taking the best possible policy from the learned coverage set compared to the one from the optimal coverage set:

$$MUL(\Pi) = \max_{u \in \mathcal{U}} \left(\max_{\pi^* \in \Pi^*} u(\boldsymbol{V}^{\pi^*}) - \max_{\pi \in \Pi} u(\boldsymbol{V}^{\pi}) \right). \tag{2.21}$$

While EUM indicates the average performance of a solution set, MUL provides information about the worst-case scenario. The advantage of measuring worst-case performance is that we only require to know the set $\mathcal{U}$, not its probability distribution $\mathcal{P}(\cdot \mid \mathcal{U})$. However, like EUL, this metric requires Π^*, which is not always available.

2.8.5 The ε-indicator

EUM, EUL and MUL are metrics that correlate with the user's utility. We believe these are most relevant, since the end-goal under SER or ESR is to maximize the user utility. However, all three metrics require knowledge over the space $\mathcal{U}$, which we do not necessarily know. As an alternative, we could apply the same principles as the previous metrics, but on the objectives themselves instead of the utility. This can then serve as a proxy for the utility-based metrics. An alternative metric to the MUL that does not require $\mathcal{U}$ is the *ε-indicator* I_ε (Vamplew et al., 2017; Zitzler et al., 2003).

I_ε measures how close a coverage set is to the Pareto front P. The ε-indicator of a coverage set $\hat{\Pi}$ is computed such that, for every $\boldsymbol{V}^\pi$ of the Pareto front, there exist a V-value in the coverage set that is at most ϵ smaller than $\boldsymbol{V}^\pi$:

$$I_\varepsilon = \inf_{\varepsilon \in \mathbb{R}} \{\forall \boldsymbol{V}^\pi \in P,\ \exists \boldsymbol{V}^{\pi'} \in \hat{\Pi} :\ ||V^\pi - V_o^{\pi'}||_\infty \leq \varepsilon\}. \tag{2.22}$$

From a user's perspective, I_ε has the intuitive meaning of showing that the proposed coverage set is at most ε worse than any V-value of the Pareto front. Thus, just as for MUL, this is a worst-case performance metric. Also, like MUL, the main disadvantage of this metric is that to compute it we need the true Pareto front. Again, in practice we can approximate the Pareto front using the non-dominated policies across all our experiments.

2.8.6 The $\varepsilon - mean$-indicator

While the ε-indicator has nice theoretical properties guaranteeing performance in the worst case (Zintgraf et al., 2015) — i.e., the MUL with respect to any possible utility function — it does not give any information about the EUL. This makes it a highly pessimistic metric; the ε-indicator will still report bad performance even if nearly the whole Pareto front is learned exactly, as long as a single point is not correctly modeled.

While measuring the EUL precisely requires knowing in advance the distribution of possible utility functions, we propose a variation on the I_ε metric that aims to approximate an upper bound to this value, by making an additional assumption (Reymond, Eugenio, & Nowè, 2022). In particular, we propose computing $I_{\varepsilon-mean}$, that assumes that each point in the Pareto front is equally likely to be selected as the best choice by a randomly sampled utility function. We compute $I_{\varepsilon-mean}$ by taking the mean of the computed ε values,

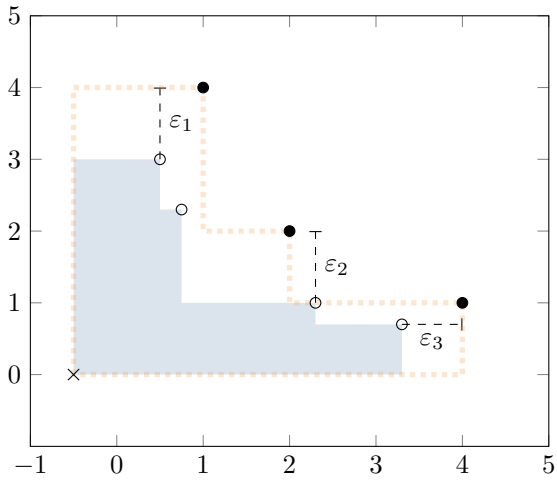


Figure 2.3: Pareto front and coverage set in 2-objective environment. The Pareto front is the set containing the best compromises that can be achieved (black dots). We want to learn a coverage set that is as close as possible to the true Pareto front (white dots). The hypervolume metric, in light blue, measures the volume of all dominated solutions w.r.t. some reference point (cross). The ε metrics first compute the maximum distance between each point in the Pareto front and its closest point in the coverage set (ε_i). We can then take their maximum value to compute the I_ε metric, or their mean value to obtain the $I_{\varepsilon-mean}$ metric of the coverage set.

which are computed in the same way as the original I_ε metric. Because the MUL of incorrectly preferring another vector in the coverage set over a given vector in the Pareto front is $\varepsilon\sqrt{n}L$ (Zintgraf et al., 2015), where L is the Lipschitz constant that described the level of continuity of the utility function, the $I_{\varepsilon-mean}$ metric describes an upper bound on the EUL, given that each vector in the Pareto set is (approximately) equally likely to be the vector that maximizes the user's utility. The hypervolume and ε-indicator are depicted in Figure 2.3.

Chapter 3

Incorporating non-linear utility functions to learn the optimal policy under ESR

3.1 The challenge of the ESR optimization criterion 30
3.2 Using accrued rewards to compute the utility 31
3.2.1 Markov property for MOMDPs with accrued rewards 31
3.3 Optimizing the utility using single-objective methods 31
3.3.1 Mapping function for linear scalarization functions 32
3.4 Policy gradient for multi-objective reward functions 33
3.5 Using a distributional critic to estimate the future returns 35
3.6 Multi-Objective Categorical Actor-Critic 36
3.6.1 Multi-objective policy gradient with future return estimates 37
3.6.2 Incorporating a distributional critic 39
3.6.3 Corollary: A multi-objective actor for SER 41
3.7 Ablation study 43
3.7.1 Using a distributional critic 43
3.7.2 Conditioning on accrued rewards 44
3.7.3 Illustration on general monotonically increasing utility functions . 46
3.8 Benchmark environments 48
3.8.1 Deep-Sea-Treasure 48
3.8.2 Minecart 49
3.9 Experiments 50
3.9.1 Deep-Sea-Treasure 51
3.9.2 Minecart 53
3.10 Related work 55
3.11 Discussion 56

3.1 The challenge of the ESR optimization criterion

In Section 1.2 we motivate how we can classify Multi-objective reinforcement learning (MORL) algorithms in 3 main categories, depending on the knowledge we have over the utility function u. In this chapter, we assume we are able to extract this utility function before running the learning algorithm, e.g., by using preference elicitation methods (Zintgraf et al., 2018). The utility function can thus be used to find the policy that leads to the preferred outcome.

However, when user preferences are non-linear – as human preferences often are – the utility function cannot be used to straightforwardly reduce a multi-objective problem to a single-objective one. Dedicated multi-objective models and methods are required, even if the utility function is known a priori (Roijers et al., 2013). After all, reinforcement learning (RL) relies heavily on the assumption that optimizing the (discounted) sum of individual rewards will also optimize the overall problem. But with non-linear utility functions, the sum of the utility function applied to the individual rewards is *not* equal to the utility over the return:

$$u\left(\sum_{t=0}^{h} \gamma^t \boldsymbol{r}_t\right) \neq \sum_{t=0}^{h} \gamma^t u\left(\boldsymbol{r}_t\right). \tag{3.1}$$

This makes our setting a hard-to-solve problem, as the vast majority of RL algorithms cannot be straightforwardly applied: they use the Bellman equation, which takes advantage of this sum-of-rewards assumption. Moreover, most work on MORL applies the utility function on the expected returns (the Value $\boldsymbol{V}$), not directly on the discounted sum of rewards (the return $\boldsymbol{R}$). That is to say, the vast majority of MORL algorithms optimize using the SER criterion, not the ESR optimization criterion.

We focus on the latter, which implies that each policy evaluation is relevant to the user in terms of utility. This also means that the utility function cannot be applied on the Values, as they estimate expected returns. Perhaps this is why, even though identified as an open challenge in the seminal survey on MORL (Roijers et al., 2013), this setting is still severely understudied in the MORL literature, most bodies of work instead bypassing this issue by assuming that the utility function is a weighted sum over the objectives.

In this chapter, we overcome this challenge by proposing a novel algorithm that explicitly keeps track of the different objectives and correctly applies the non-linear utility function on estimates of the overall multi-objective return. Our key insight is that, if the agent learns a multi-variate distribution over the future returns, we can use this distribution to bootstrap, enabling us to exploit the Bellman equation in MORL. Using this insight, we propose an actor-critic method we call *Multi-Objective Categorical Actor-Critic* (MOCAC) (Reymond, Hayes, et al., 2023), that uses such bootstrapping in its critic. We implement and demonstrate our methods on multiple multi-objective benchmarks, and show that they learn effectively where single-objective baselines fail. To the best of our knowledge, this is the first MORL algorithm that exploits a priori known non-linear utility functions to optimize the expected utility.

3.2 Using accrued rewards to compute the utility

In reinforcement learning, the optimal policy is the one that maximizes the expected return. For MORL, the maximization is done with respect to the utility on the return. This means the utility function can only be applied on whole episodic returns. Thus, at any timestep t, it is essential to not only estimate the future rewards, but also take into account the accrued rewards: the rewards accumulated from timestep 0 until timestep t.

Consider a 2-objective MOMDP where, at timestep $t = T - 1$, the agent has already accumulated a total reward $\boldsymbol{R}_t^- = (5, 0)$. The preferences of the decision maker are formalized as the non-linear utility function $u = \min(r_0, r_1)$ (i.e., the objectives are perfect complements). Let us say the agent has two possible actions, each leading it to a final state, thus ending the episode. The first action a_0 results in reward $\boldsymbol{r}_t = (2, 2)$, while the second action a_1 results in reward $\boldsymbol{r}_t = (0, 5)$. If only future rewards are considered, executing a_0 will result in $u = 2$, while a_1 will result in $u = 0$ (a_0 is optimal). However, if we take into account the accrued rewards, a_0 will result in a total episodic return of $(7, 2)$ ($u = 2$), while a_1 will result in $(5, 5)$ ($u = 5$). In this case, the action leading to the highest user utility is a_1. We thus propose to incorporate the accrued reward in order to make correct use of u. Given, at timestep t, the future discounted return $\boldsymbol{R}_t = \sum_{k=t}^{H} \gamma^{k-t} \boldsymbol{r}_k$, we define the accrued reward as:

$$\boldsymbol{R}_t^- = \sum_{k=0}^{t-1} \gamma^k \boldsymbol{r}_k. \tag{3.2}$$

The episodic return is then simply the sum of the accrued rewards and $\boldsymbol{R}_t$:

$$\boldsymbol{R} = \boldsymbol{R}_t^- + \gamma^t \boldsymbol{R}_t. \tag{3.3}$$

3.2.1 Markov property for MOMDPs with accrued rewards

By incorporating the accrued reward into the utility function, we can directly optimize the policy on the user utility. Note that by using a history of past rewards to update a policy conditioned on state s_t we break the Markov property. This, however, can be prevented by conditioning the state on the accrued rewards. Thus, we can transform any MOMDP $\mathfrak{M}_1 = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \boldsymbol{\mathcal{R}}, n \rangle$ to another MOMDP $\mathfrak{M}_2 = \langle \mathcal{S} \times \mathbb{R}^n, \mathcal{A}, \mathcal{T}, \gamma, \boldsymbol{\mathcal{R}}, n \rangle$ such that, at any timestep t, we can keep track of the reward that has been accumulated since the start-state s_0.

3.3 Optimizing the utility using single-objective methods

Although single-objective methods are not applicable to MOMDPs, since we assume the utility function is known we can optimize on the scalar utility using accrued rewards. We propose a mapping function that, given a MOMDP and a utility function, returns a MDP for which the optimal policy is the same, i.e., $f_{\text{end}} \colon \mathfrak{M} \times u \to \mathfrak{M}$.

Formally, given $\mathfrak{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \boldsymbol{\mathcal{R}}, n \rangle$, we define $\mathfrak{M} = \langle \mathcal{S} \times \mathbb{R}^n, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R} \rangle$ as the single-objective MDP with the same optimal policy, where $\mathfrak{M}$'s state-space is augmented following Section 3.2.1 to guarantee the Markov property, the action-space $\mathcal{A}$ and discount factor γ are the same as for $\mathfrak{M}$, and the reward function returns a zero-reward in every state, except when the agent reaches a terminal state, in which case the reward function returns the utility of the accrued rewards, i.e., the utility of the episodic return:

$$\mathcal{R}(s, a, s') = \begin{cases} u(\boldsymbol{R}_T^- + \gamma^T \boldsymbol{\mathcal{R}}(s, a, s')) & \texttt{if} \;\; s' \in \mathcal{S}_f \\ 0 & \text{otherwise}, \end{cases} \tag{3.4}$$

where $\mathcal{S}_f \subseteq \mathcal{S}$ is the set of final states. We call this mapping the *terminal mapping*.

Two issues arise with this transform. Firstly, it reduces the problem to a single-objective MDP with a highly sparse reward function. Such sparse problems are notoriously hard to learn, especially for long episodes (a well-known example is the Atari 2600 game *Montezuma's Revenge* (Burda et al., 2019)).

Secondly, as with accrued rewards, this transform breaks the Markov property. To cope with this issue, we can augment the state-space with the accrued rewards, resulting in a state-space of size $\mathcal{S} \times \mathbb{R}^n$. Since the state-space is increased compared to the original MOMDP, we expect that RL algorithms will need an increased number of interactions with the environment, as more exploration is required to find the optimal policy.

This increased exploration problem is applicable for any MOMDPs with non-linear utility functions, since it is then mandatory to keep track of the accrued rewards to retain the Markov property. However, it is exacerbated for the terminal scalarization due to the sparsity of rewards. Learning the optimal policy for MDPs with sparse rewards is hard, as we typically need to explore a large part of the state-space before reaching higher returns. Since the state-space itself is augmented, even more exploration is required. We show in Section 3.9 that, while it is possible to use single-objective algorithms with this transform, it is inefficient to do so, as many interactions with the environments are required. Dedicated multi-objective approaches take advantage of the multiple reward signals, removing the sparse reward problem and thus alleviating the exploration issue.

As a final caveat, we note that, for infinite time-horizons with no terminal states, the reward is multi-objective reward is accumulated indefinitely and the single-objective reward perceived by the agent is always zero. Thus, this transform assumes a finite time-horizon setting.

3.3.1 Mapping function for linear scalarization functions

The terminal mapping results in a sparse reward function. Assuming we know the utility function is a linear scalarization, we can take advantage of its additivity property (Equation 2.7) to alleviate this issue. Since, in this case, the sum of the utilities of individual rewards is equal to the utility of the episodic return, we propose a mapping $f_{\text{step}} \colon \mathfrak{M} \times u \rightarrow \mathfrak{M}$, in which $\mathfrak{M}$'s reward function returns, at every timestep, the utility of the vectorial reward returned by $\mathfrak{M}$'s reward function. Thus, given $\mathfrak{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \boldsymbol{\mathcal{R}}, n \rangle$, the mapping function f_{step} results in $\mathfrak{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R} \rangle$, with:

$$\mathcal{R}(s, a, s') = u(\boldsymbol{\mathcal{R}}(s, a, s')). \tag{3.5}$$

We call this mapping the *timestep* mapping. We use it as a baseline in our experimental section to show that, when the utility function is non-linear, the optimal policy resulting from applying u at every timestep does not maximize the decision maker's utility.

3.4 Policy gradient for multi-objective reward functions

As mentioned above, no existing MORL algorithm can cope with our setting. Therefore, we develop a novel algorithm based on the classic policy gradient algorithm, Reinforce (Williams, 1992), to multi-objective optimization. Reinforce makes use of the Policy Gradient theorem to provide convergence guarantees of the policy towards a local optimum via gradient ascent. We aim to provide the same convergence guarantees for MOMDPs.

Given a performance measure $J(\pi_\theta)$, with θ the parameters of the policy π, we would like to use its gradient $\nabla_\theta J(\pi_\theta)$ to update the policy using gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k})$$

where α is the learning rate. For single-objective RL, the performance measure is the expected return:

$$J(\pi_\theta) \doteq \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} [R(\tau)] . \tag{3.6}$$

To derive its gradient, we use the log-derivative trick:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x} \log f(x) &= \frac{1}{f(x)} \frac{\mathrm{d}}{\mathrm{d}x} f(x) \\ \frac{\mathrm{d}}{\mathrm{d}x} f(x) &= f(x) \frac{\mathrm{d}}{\mathrm{d}x} \log f(x) \end{aligned} \tag{3.7}$$

The gradient $\nabla_\theta J(\pi_\theta)$ of $J(\pi_\theta)$ is derived as follows:

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] \\
&= \nabla_\theta \int_\tau \mathcal{P}(\tau \mid \pi_\theta) R(\tau) \\
&= \int_\tau \nabla_\theta \mathcal{P}(\tau \mid \pi_\theta) R(\tau) \\
&\overset{3.7}{=} \int_\tau \mathcal{P}(\tau \mid \pi_\theta) \nabla_\theta \log \mathcal{P}(\tau \mid \pi_\theta) R(\tau) \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\nabla_\theta \log \mathcal{P}(\tau \mid \pi_\theta) R(\tau)] \\
&= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)\right]. \qquad (3.8)
\end{aligned}
$$

This expectation can be estimated using a sample mean, after having collected a set of trajectories using π_θ, resulting in the Reinforce algorithm:

Algorithm 1 Reinforce (Williams, 1992)

Require: Parametric policy π_θ, learning rate α

```
while 1 do
    Collect set of trajectories {τ_i}_{i=1}^N using π_τ
    Estimate ∇_θ J(π_θ) with Σ_{i=1}^N Σ_{t=0}^T ∇_θ log π_θ(a_t | s_t) R(τ_i)
    θ ← θ + α ∇_θ J(π_θ)
end while
```

Reinforce is an actor-only method, as it directly optimizes the policy, and does not estimate the policy's Value. Since it collects a set of trajectories at each iteration, the policy is updated at most once per episode (when the size of the set is one). This makes it a quite sample-inefficient algorithm. However, this has the advantage of being able to compute the full discounted return which, for ESR optimization, is necessary to compute the utility. Thus, we propose a multi-objective variant of Reinforce called MO Reinforce[1], that optimizes the user utility under ESR, by adapting Equation 3.8 to vectorial rewards, using the ESR optimization criterion as performance measure:

$$
J(\pi_\theta) \doteq \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[u(\boldsymbol{R}(\tau))]. \qquad (3.9)
$$

[1]We note that our preliminary workshop paper (Roijers et al., 2018) we referred to this algorithm as expected utility policy gradient (EUPG).

This results in:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[u(\boldsymbol{R}(\tau))] \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t) u(\boldsymbol{R}(\tau))\right] \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t) u(\boldsymbol{R}_t^- + \boldsymbol{R}_t)\right]. \end{aligned} \tag{3.10}$$

In the last step of Equation 3.10, we explicitly show the use of accrued rewards, as a reminder that augmenting the state-space with the accrued rewards (as explained in Section 3.2) is necessary to keep the Markov property.

Using our proposed MO Reinforce, we can directly optimize the policy for ESR, even when the utility function is non-linear. However, as for the original Reinforce algorithm, MO Reinforce updates its policy once after each episode, resulting in a sample-inefficient algorithm.

3.5 Using a distributional critic to estimate the future returns

By using the Policy Gradient theorem, we can directly optimize the policy on the episodic return. However, many interactions with the environment are required before being able to update the policy, as we have to execute the policy until the end of the episode to compute the return. If, instead, at each timestep t, we could estimate the future return R_t, we would not have to wait until the whole episode is played out before updating the policy, as we could then replace the return in Equation 3.8 with our estimate, and update the policy at the current timestep based on this estimate. This is the core idea behind the actor-critic framework, which has been used to produce many state-of-the-art deep reinforcement learning algorithms (Abdolmaleki et al., 2018; Fujimoto et al., 2018; Haarnoja et al., 2018; Mnih et al., 2016; Schulman et al., 2017). At its essence, the actor-critic framework leverages two components. On the one hand, the actor learns the policy, a probability distribution over the actions in $\mathcal{A}$, conditioned on a given state. These probabilities are updated regularly, increasing the probabilities of actions that lead to high returns by using the negative log-likelihood loss. On the other hand, the critic learns to estimate the future expected returns for a given state-action pair. This estimate is commonly known as the Q-value:

$$Q(s,a) = \mathbb{E}\left[\sum_{k=0}^{h}\gamma^k r_{t+k} \mid \pi, s_t = s, a_t = a\right]. \tag{3.11}$$

Note that the Q-value is similar to the Value, the difference being that the Q-value is conditioned on both the state and the action, while the Value is conditioned on the state only.

Using a critic allows the actor to be updated at every timestep, by estimating the future expected returns using the critic instead of waiting for them to be played out. However, the goal of our multi-objective setting is to learn a policy that leads to the highest user utility for every single evaluation. Thus, u needs to be applied on the episodic return, not on the expected return. In light of this, we incorporate distributional reinforcement learning into our algorithm, by creating a critic that estimates a multi-variate distribution $\mathcal{Z}^\pi$ over return values, inspired by the work of Bellemare et al. (2017).

In single-objective RL, the expected return from a state s_t, $V^\pi(s_t)$, can be decomposed into a distribution $\mathcal{Z}^\pi$ over future returns:

$$V^\pi(s_t) = \int_R R\mathcal{Z}^\pi(R \mid s_t), \tag{3.12}$$

where $\mathcal{Z}(R \mid s_t)$ represents the probability of having a return R from state s_t. Learning the whole distribution instead of just the expected return leads to more stable learning. To solve our multi-criteria setting, we build upon this idea, and extend it to the multivariate case, allowing us to learn the distribution over n-dimensional future returns. In contrast to single-objective learning, this is actually essential for MORL under ESR as we need to sum the accrued return with future returns *before* applying the utility function and taking the expectation. Without employing a distributional critic, we would not be able to do so. Indeed, under the ESR optimization criterion, we can use a multivariate $\boldsymbol{\mathcal{Z}}$ to express the optimal policy as follows:

$$\pi^* = \arg\max_\pi \int_{\boldsymbol{R}} u(\boldsymbol{R})\boldsymbol{\mathcal{Z}}^\pi(\boldsymbol{R} \mid s_0). \tag{3.13}$$

Moreover, we can estimate the utility for a state s_t by incorporating the accrued returns:

$$V^\pi_{u,\text{ESR}}(s_t) = \int_{\boldsymbol{R}} u(\boldsymbol{R}_t^- + \boldsymbol{R})\boldsymbol{\mathcal{Z}}^\pi(\boldsymbol{R} \mid s_t, \boldsymbol{R}_t^-). \tag{3.14}$$

Using this insight, we propose a novel multi-objective algorithm that incorporates this multivariate distributional critic to optimize the policy under the ESR criterion.

3.6 Multi-Objective Categorical Actor-Critic

We introduce *Multi-Objective Categorical Actor-Critic* (MOCAC), our proposed algorithm for ESR under known non-linear utility functions. MOCAC is based on the actor-critic framework. First, we demonstrate how the actor optimizes the decision maker's utility using a multivariate $\boldsymbol{\mathcal{Z}}$ to estimate the future returns. Then, we explain how the critic is used to learn $\boldsymbol{\mathcal{Z}}$.

3.6.1 Multi-objective policy gradient with future return estimates

Equation 3.10 shows that we can optimize the policy (or actor) based on the multi-objective returns. We now show that we can replace this return by an estimate, and still optimize the policy with respect to the same performance indicator, i.e., the ESR criterion.

Starting from:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) u(\boldsymbol{R}(\tau))\right],$$

using the law of iterated expectations, we split the trajectory τ into two parts: $\tau_{:t}, \tau_{t:}$ the parts of the trajectories before timestep t and from t onwards, respectively. We also use our definition of accrued rewards (Section 3.2) to split the episodic return $\boldsymbol{R}(\tau)$ into the accrued return $\boldsymbol{R}_t^-$ and future return $\boldsymbol{R}_t$.

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) u(\boldsymbol{R}(\tau))\right] \\
&= \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) u(\boldsymbol{R}(\tau))\right] \\
&= \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau_{:t}\sim\pi_\theta}\left[\mathop{\mathbb{E}}_{\tau_{t:}\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) u(\boldsymbol{R}(\tau)) \mid \tau_{:t}\right]\right] \\
&= \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau_{:t}\sim\pi_\theta}\left[\mathop{\mathbb{E}}_{\tau_{t:}\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) u(\boldsymbol{R}_t^- + \boldsymbol{R}_t) \mid \tau_{:t}\right]\right] \qquad (3.15)
\end{aligned}$$

Since $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ is constant with respect to the inner expectation, we can take it out:

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau_{:t}\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \mathop{\mathbb{E}}_{\tau_{t:}\sim\pi_\theta}\left[u(\boldsymbol{R}_t^- + \boldsymbol{R}_t) \mid \tau_{:t}\right]\right]. \qquad (3.16)$$

Because we augment the state-space with the accrued reward, $\boldsymbol{R}_t^-$ depends only on s_t. Moreover, due to the Markovian property of (MO)MDPs, the future return $\boldsymbol{R}_t$ only depends on s_t, a_t. Given that $\boldsymbol{R}_t^-$ depends only on s_t and $\boldsymbol{R}_t$ only on s_t, a_t, $u(\boldsymbol{R}_t^- + \boldsymbol{R}_t)$ does not depend on the past:

$$\mathop{\mathbb{E}}_{\tau_{t:}\sim\pi_\theta}\left[u(\boldsymbol{R}_t^- + \boldsymbol{R}_t) \mid \tau_{:t}\right] = \mathop{\mathbb{E}}_{\tau_{t:}\sim\pi_\theta}\left[u(\boldsymbol{R}_t^- + \boldsymbol{R}_t) \mid s_t, a_t\right]. \qquad (3.17)$$

Since a MOMDP is composed of a discrete number of states and actions, there are a finite number of possible returns $\boldsymbol{R}$. Thus, expanding the expectation of the previous equation results in:

$$\mathop{\mathbb{E}}_{\tau_{t:}\sim\pi_\theta}\left[u(\boldsymbol{R}_t^- + \boldsymbol{R}_t) \mid s_t, a_t\right] = \sum_{\boldsymbol{R}} u(\boldsymbol{R}_t^- + \boldsymbol{R}) \mathcal{Z}(\boldsymbol{R} \mid s_t, a_t), \qquad (3.18)$$

where $\boldsymbol{\mathcal{Z}}(\boldsymbol{R} \mid s_t, a_t)$ is the probability of obtaining future return R from s_t, a_t. The policy gradient is thus:

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau_{:t} \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{\boldsymbol{R}} u(\boldsymbol{R}_t^- + \boldsymbol{R}) \boldsymbol{\mathcal{Z}}(\boldsymbol{R} \mid s_t, a_t) \right]. \tag{3.19}$$

Improving learning stability using a baseline

At timestep t, we thus use $\boldsymbol{\mathcal{Z}}$ to estimate the utility of the episode, and update the policy accordingly. The policy is therefore potentially updated at each timestep, making it sample-efficient compared to MO Reinforce.

In practice, the expectation of Equation 3.19 is estimated using a sample mean, after having collected a set of trajectories $\{\tau_i\}_{i=1}^N$ using π_θ. However, depending on the reward function, the return can greatly vary between episodes, potentially resulting in a high variance estimate. We now aim to reduce this variance to improve the stability of the learning process.

For any function $b(s_t)$ that only depends on the state s_t (and not on the action a_t), we have the property that:

$$\begin{aligned}
\mathop{\mathbb{E}}_{a_t \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a_t \mid s_t) b(s_t)] &= \mathop{\mathbb{E}}_{a_t \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a_t \mid s_t)]\, b(s_t) \\
&= \sum_{a \in \mathcal{A}} \pi_\theta(a_t \mid s_t) \nabla_\theta \log \pi_\theta(a_t \mid s_t) b(s_t) \\
&\overset{3.7}{=} \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a_t \mid s_t) b(s_t) \\
&= \nabla_\theta \sum_{a \in \mathcal{A}} \pi_\theta(a_t \mid s_t) b(s_t) = \nabla_\theta 1 b(s_t) = 0.
\end{aligned} \tag{3.20}$$

We call $b(s_t)$ a *baseline* (Mnih et al., 2016). Interestingly, this equation shows that the baseline has the property of not affecting the expectation, if added in the policy gradient. We can use the exact same proof as for the original policy gradient with baseline theorem to modify Equation 3.19:

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau_{:t} \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \mathop{\mathbb{E}}_{\tau_{t:} \sim \pi_\theta} \left[(u(\boldsymbol{R}_t^- + \boldsymbol{R}_t) - b(s_t)) \mid s_t, a_t \right] \right] \\
&= \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau_{:t} \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \mathop{\mathbb{E}}_{\tau_{t:} \sim \pi_\theta} \left[u(\boldsymbol{R}_t^- + \boldsymbol{R}_t) \mid s_t, a_t \right] - \right. \\
&\quad \left. \nabla_\theta \log \pi_\theta(a_t \mid s_t) \mathop{\mathbb{E}}_{\tau_{t:} \sim \pi_\theta} \left[b(s_t) \mid s_t, a_t \right] \right] \\
&= \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau_{:t} \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \mathop{\mathbb{E}}_{\tau_{t:} \sim \pi_\theta} \left[(u(\boldsymbol{R}_t^- + \boldsymbol{R}_t)) \mid s_t, a_t \right] - 0 \right].
\end{aligned} \tag{3.21}$$

Instead of using the return, which greatly affects the variance of the sample mean estimate of $\mathbb{E}_{\tau_{t:}\sim\pi_\theta}$, we can thus use the difference between the return and a baseline function. This is called the *Advantage* (Mnih et al., 2016). In the case of MOCAC, we use the expected utility for state s_t as baseline, i.e. $b(s_t) = \sum_{\boldsymbol{R}} u(\boldsymbol{R}_t^- + \boldsymbol{R})\mathcal{P}(\boldsymbol{R} \mid s_t)$. Plugging this in Equation 3.19 results in:

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\tau_{:t}\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) A(s_t, a_t)\right], \tag{3.22}$$

with

$$A(s_t, a_t) = \sum_{\boldsymbol{R}} u(\boldsymbol{R}_t^- + \boldsymbol{R})\mathcal{P}(\boldsymbol{R} \mid s_t, a_t) - \sum_{\boldsymbol{R}} u(\boldsymbol{R}_t^- + \boldsymbol{R})\mathcal{P}(\boldsymbol{R} \mid s_t). \tag{3.23}$$

We use this gradient to update the policy of MOCAC with gradient ascent.

3.6.2 Incorporating a distributional critic

Next to the actor, the second component of MOCAC is the critic. We have explained in Section 3.6.2 that we can use a distributional critic to optimize under the ESR criterion. We take inspiration from Bellemare et al. (2017), who propose a distributional variant of Deep Q-Networks (DQN) (Mnih et al., 2015).

Bellemare et al. (2017) use a parametric discrete distribution $\mathcal{Z}_\psi$ to approximate $\mathcal{Z}^\pi$ as it is computationally friendly and highly representative. We arbitrarily choose the number of categories C of $\mathcal{Z}_\psi$. Each category $i \in [C]$ of $\mathcal{Z}_\psi$ is defined as a support atom $z_i = V_{\text{MIN}} + i\Delta z$, with $\Delta z := \frac{V_{\text{MAX}} - V_{\text{MIN}}}{C-1}$, where $V_{\text{MIN}}, V_{\text{MAX}} \in \mathbb{R}$ represent the smallest and largest return values of the distribution, respectively. Note that V_{MIN} and V_{MAX} are bounds on the returns, not on the value, but we use the V-notation to remain consistent with Bellemare et al. (2017). Thus, $Z_\psi \colon \mathcal{S} \times N \to [0, 1]$ is a probabilistic function which represents the probability of ending with a return $R_t \in [z_i, z_{i+1})$, given a state s_t and a category i.

For our algorithm, we require a multivariate distribution where $\boldsymbol{z}_i$ is a support atom for vectorial returns. We provide a separate $V_{\text{MIN}}, V_{\text{MAX}}$ for each objective. Moreover, we assume the same number of categories C for each objective-dimension, resulting in a discrete distribution with C^n categories. Each atom $\boldsymbol{z}_i \; \forall i \in [C]^n$ then becomes:

$$\begin{aligned} \boldsymbol{z}_i &= \boldsymbol{V}_{\text{MIN}} + i\Delta\boldsymbol{z}, \\ \text{with } \Delta\boldsymbol{z} &:= \frac{\boldsymbol{V}_{\text{MAX}} - \boldsymbol{V}_{\text{MIN}}}{C-1}. \end{aligned} \tag{3.24}$$

$\boldsymbol{V}(s_t)$ is computed in a similar manner as the single-objective case (Equation 3.12):

$$\boldsymbol{V}(s_t) = \sum_{i \in [C]^n} \boldsymbol{z}_i \boldsymbol{\mathcal{Z}}_\psi(\boldsymbol{z}_i \mid s_t).$$

Because the critic now represents a full distribution instead of an expected value, we overcome the second challenge of our setting: since $\boldsymbol{z}_i$ is defined as a return with its associated probability $\boldsymbol{\mathcal{Z}}_\psi(\boldsymbol{z}_i)$, it can be converted into a preference score using u under the ESR criterion. This results in the following equation:

$$u_t = \sum_{i \in [C]^n} u(\boldsymbol{R}_t^- + \gamma^t \boldsymbol{z}_i) \boldsymbol{\mathcal{Z}}_\psi(\boldsymbol{z}_i \mid s_t). \tag{3.25}$$

Note that we include the accrued reward as defined in Section 3.2 to correctly compute the utility.

Updating the critic

To update the critic, we compute the distributional Bellman update $\hat{\mathcal{T}} \boldsymbol{z}_j := \boldsymbol{r}_t + \gamma \boldsymbol{z}_j$ for each atom $\boldsymbol{z}_j$, for a given sample transition $(s_t, a_t, \boldsymbol{r}_t, s_{t+1})$. We then distribute its probability $\boldsymbol{\mathcal{Z}}_\psi(\boldsymbol{z}_j \mid s_t)$ to the immediate neighbors of $\hat{\mathcal{T}} \boldsymbol{z}_j$. Each of the components of the projected update is:

$$(\Phi \hat{\mathcal{T}} \boldsymbol{\mathcal{Z}}_\psi(s_t))_i = \sum_j \left[1 - \frac{|[\hat{\mathcal{T}} \boldsymbol{z}_j]_{\boldsymbol{V}_{\text{MIN}}}^{\boldsymbol{V}_{\text{MAX}}} - \boldsymbol{z}_i|}{\Delta \boldsymbol{z}} \right]_0^1 \boldsymbol{\mathcal{Z}}_\psi(\boldsymbol{z}_i \mid s_{t+1}), \tag{3.26}$$

with $[.]_b^a$ bounding the argument between $[a, b]$.

We use the cross-entropy term of the KL-divergence as the loss function for the critic:

$$D_{KL}(\Phi \hat{\mathcal{T}} \boldsymbol{\mathcal{Z}}(s) || \boldsymbol{\mathcal{Z}}(s)). \tag{3.27}$$

Thus, we propose Multi-Objective Categorical Actor-Critic (MOCAC), an algorithm that optimizes the utility under the ESR criterion and is able to take advantage of any kind of monotonically increasing utility function. The algorithm is summarized in Algorithm 2. To the best of our knowledge, it is the first reinforcement learning algorithm to cope with this setting. Moreover, we show in the experimental section that it is also stable and sample-efficient.

Algorithm 2 Multi-Objective Categorical Actor-Critic

Require: Policy, critic parameters $\theta.\psi$, number of categories C, maximal and minimal possible return $\boldsymbol{V}_{\text{MAX}}, \boldsymbol{V}_{\text{MIN}}$, actor, critic learning rates α_1, α_2

$t \leftarrow 0$ ▷ Start new episode
while 1 **do**
  Take action $a_t \sim \pi_\theta(s_t)$, get $s_{t+1}, \boldsymbol{r}_t$
  $u_t = \sum_{i \in [C]^n} u(\boldsymbol{R}_t^- + \gamma^t \boldsymbol{r}_t + \gamma^{t+1} \boldsymbol{z}_i) \boldsymbol{\mathcal{Z}}_\psi(\boldsymbol{z}_i \mid s_{t+1})$ ▷ Expected utility based on next-state estimate
  $b_t = \sum_{i \in [C]^n} u(\boldsymbol{R}_t^- + \gamma^t \boldsymbol{z}_i) \boldsymbol{\mathcal{Z}}_\psi(\boldsymbol{z}_i \mid s_t)$ ▷ Baseline based on current-state estimate
  $d\theta \leftarrow \nabla_\theta \log \pi_\theta(a_t \mid s_t)(u_t - b_t)$ ▷ Eq. 3.22
  $m_c = 0, \quad c \in [C]^n$
  for $c \in [C]^n$ **do**
    $\hat{\mathcal{T}} \boldsymbol{z}_c \leftarrow \boldsymbol{r}_t + \gamma \boldsymbol{z}_c$ ▷ Compute projection onto support atom
    $\boldsymbol{b}_c \leftarrow (\hat{\mathcal{T}} \boldsymbol{z}_c - \boldsymbol{V}_{\text{MIN}})/\Delta \boldsymbol{z}$ ▷ $\boldsymbol{b}_c \in [C]^n$
    for each neighbor j of $\boldsymbol{b}_c$ **do** ▷ Considering upper, lower results in 2^n neighbors
      $m_j \leftarrow m_j + \boldsymbol{\mathcal{Z}}_\psi(s_{t+1}) |\boldsymbol{b}_c - j|$
    end for
  end for
  $d\psi \leftarrow \nabla_\psi \sum_{c \in [C]^n} m_c \boldsymbol{\mathcal{Z}}_\psi(s_t)$
  $\theta \leftarrow \theta + \alpha_1 d\theta$ ▷ Gradient ascent to maximize $J(\pi_\theta)$
  $\psi \leftarrow \psi - \alpha_2 d\psi$ ▷ Gradient descent to minimize cross-entropy
  $t \leftarrow t + 1$
end while

3.6.3 Corollary: A multi-objective actor for SER

Accrued rewards are necessary for non-linear utility functions, regardless of the optimization criterion (SER or ESR). We argue that the distributional critic is required for optimizing under ESR. As such, as a corollary, we propose a modified version of MOCAC for optimizing under SER. In this version, the utility is applied on the estimated Value to cope with the different optimization criterion. This results in two consequences. First, we change the actor loss accordingly. Second, since we do not require to know the distribution over returns $\boldsymbol{\mathcal{Z}}$, the distributional critic is replaced by the more traditional critic, that learns to estimate $\boldsymbol{V}$. We call this algorithm Multi-Objective Actor-Critic (MOAC), since it does not use the distribution $\boldsymbol{\mathcal{Z}}$.

For the actor, we derive a multi-objective variant of the policy gradient theorem, which modifies the actor-update equation. Intuitively, this loss computes the policy gradient for each objective separately and weights the gradient according to the objective's importance. The importance of each objective depends on the utility function. These weighted gradients are then summed together. More formally, for the SER optimization criterion, the

performance measure we aim to maximize is defined as follows:

$$\begin{aligned} J(\pi_\theta) &\doteq u(\mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[\boldsymbol{R}(\tau)]) \\ &= u(\boldsymbol{V}^{\pi_\theta}(s_0)). \end{aligned} \tag{3.28}$$

The gradient of the policy performance, i.e. the policy gradient, is:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \nabla_\theta u(\boldsymbol{V}^{\pi_\theta}(s_0)) \\ &= \sum_{i=0}^{n} \frac{\partial u}{\partial V_i^{\pi_\theta}} \frac{\partial V_i^{\pi_\theta}}{\partial \theta} \\ &= \sum_{i=0}^{n} \frac{\partial u}{\partial V_i^{\pi_\theta}} \nabla_\theta V_i^{\pi_\theta}(s_0), \end{aligned} \tag{3.29}$$

with $V_i^{\pi_\theta}$ the Value for the i-th objective. Since this is a scalar, $\nabla_\theta V_i^{\pi_\theta}(s_0)$ follows the original Policy Gradient proof (in our case with baseline):

$$\nabla_\theta V_i^{\pi_\theta}(s_0) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(Q_i^{\pi_\theta}(s_t, a_t) - V_i^{\pi_\theta}(s_t))\right]. \tag{3.30}$$

Intuitively, our MOAC policy gradient is a weighted sum over the individual single-objective policy gradients where each weight defines the importance of its objective, which is measured using the utility function. Concretely, the MOAC policy gradient equals to the sum of the single-objective policy gradients multiplied by the gradient of the utility function when evaluating the utility function for s_0. Since, in the setting of this chapter, u is known, we assume we can compute its derivative:

$$\nabla_\theta J(\pi_\theta) = \sum_{i=0}^{n} \frac{\partial u}{\partial V_i^{\pi_\theta}} \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)(Q_i^{\pi_\theta}(s_t, a_t) - V_i^{\pi_\theta}(s_t))\right] \tag{3.31}$$

We use this gradient to update the policy of MOAC with gradient ascent.

Concerning the critic, we update the estimate of $\boldsymbol{V}$ according to a vectorial version of temporal difference, where each objective is updated independently:

$$\boldsymbol{V}(s_t) \leftarrow \boldsymbol{V}(s_t) + \alpha(\boldsymbol{r}_t + \gamma \boldsymbol{V}(s_{t+1}) - \boldsymbol{V}(s_t)) \tag{3.32}$$

where α is the learning rate.

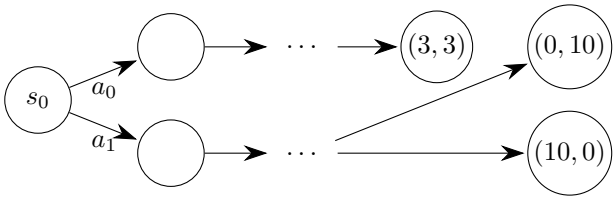


Figure 3.1: A visual representation of the Split environment. In our experiment, the length of each hallway is 10. Only zero-rewards are provided, except at the end of the hallway.

3.7 Ablation study

MOCAC uses two different components to optimize under ESR for non-linear utility functions. In this section we perform an ablation study showing that each of the components is crucial to learn the optimal policy. We first show the effects of omitting the distributional critic (see Section 3.7.1). Then, we demonstrate the importance of using accrued rewards (see Section 3.7.2). Finally, we show that MOCAC performs well on a wide range of different utility functions (see Section 3.7.3).

3.7.1 Using a distributional critic

To demonstrate the need of a distributional critic, we first perform an ablation study on a (new) simple environment we call Split. We compare our method with MOAC, the modified version of MOCAC described in Section 3.6.3, which does not use a distributional critic, but does incorporate the accrued rewards.

The Split environment, depicted in Figure 3.1, is defined as follows: in the start-state, the agent can choose between two hallways of equal length. The first one leads to a reward $(3, 3)$, while the second one leads to either a reward $(10, 0)$ or $(0, 10)$. Since Split only provides a non-zero reward in the terminal states, the future return in any non-terminal state is equal to the episodic return. Thus, Split avoids the non-stationarity issue that arises with accrued rewards (see Section 3.2), allowing us to analyze the effect of a distributional critic in isolation.

We use a synthetic utility function that multiplies both rewards together:

$$u = \max(0, r_0) \max(0, r_1). \tag{3.33}$$

As such, the utilities for reaching the first and second hallway are 9 and 0, respectively.

Results are shown in Figure 3.2. Both MOCAC and MO Reinforce learn policies that lead to the optimal utility. MO Reinforce does not use a critic, and applies u directly on the episodic return. We thus expect it to reach optimality, but more slowly than our actor-critic approach, which can benefit from the learned return distributions of the critic to improve its learning. MOAC on the other hand, learns to take the wrong hallway leading to a utility of 0. The reason behind this behavior lies in its critic. It learns the expected values of each objective rather than a distribution over the returns. These expected values are $(3, 3)$ and $(5, 5)$ for the first and second hallway, respectively. Applying the utility on the resulting V-values leads to an incorrect estimate of $u = 25$ for the second path,

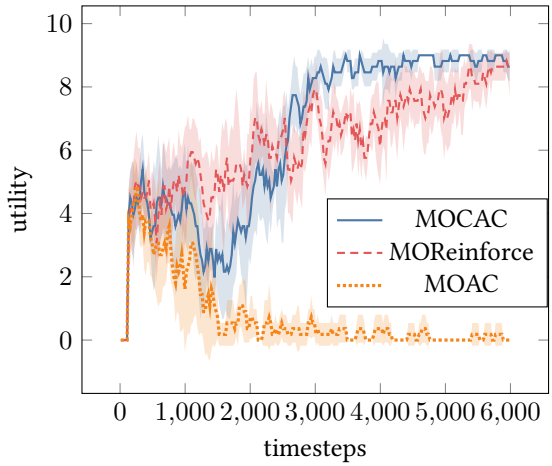


Figure 3.2: Results for the Split environment. Using a distribution over returns (MOCAC) is key for learning optimal policies when using a critic. When this is not the case (MOAC), the ESR criterion is not met.

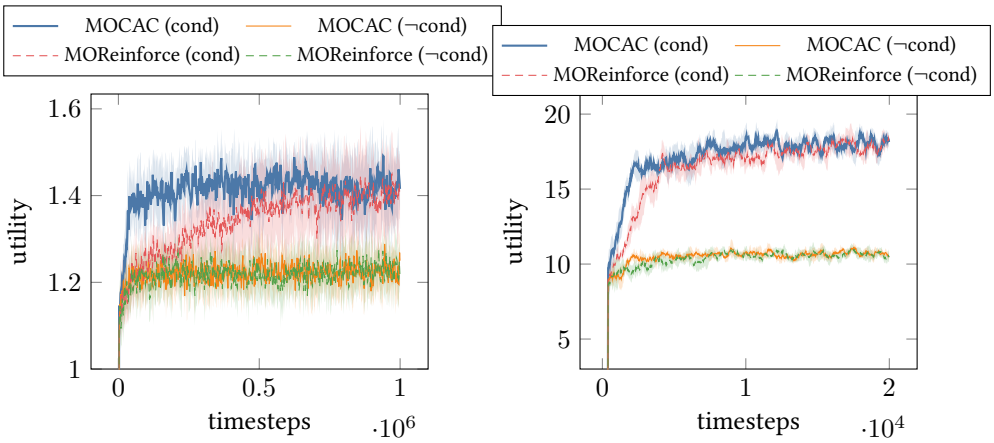
which in turn leads to corrupted advantages in for the actor gradient – and thus a poor performance.

Finally, as we can see in Figure 3.2, even though MOCAC has more parameters to learn than MO Reinforce, it reaches the optimal utility at an earlier stage, since the actor-critic algorithm makes an update at every timestep, while the baseline algorithm only updates its estimators at the end of each episode.

3.7.2 Conditioning on accrued rewards

We show in an ablation study performed on two simple MOMDPs that, depending on the setting, conditioning the policy on the accrued reward (by augmenting the state-space with the accrued reward) can be necessary to reach the optimal utility. We note that in the experiments in Section 3.9, we did not condition on the accrued rewards so far, as this turned out not to be useful in the environments from the MORL literature that we used. Nonetheless, we aim to show that conditioning can be important, using simple environments.

MiniRandom In the first environment, the agent first receives a random reward: either $(0, 5)$, $(5, 0)$ or $(0, 0)$ regardless of the action chosen, and the environment transitions to the next state. Subsequently, in this next state, it can choose between actions a_0, a_1 or a_2, which will deterministically deliver it a $(5, 0)$, $(0, 5)$ or $(2, 2)$ reward, respectively. The episode then ends. The utility function used is the same as for the Split environment, namely $u = r_0 r_1$. The optimal action to take depends on the initial random reward that the agent received. If the agent initially received $(0, 5)$, the optimal action would be the one that gives a reward $(5, 0)$, i.e., a_1, as then $u = 25$ (compared to $u = 0$ for a_0, and $u = 14$ for a_2).



(a) Results for Fishwood.

(b) Results for Mini-Random.

Figure 3.3: Comparison of MOCAC and MO Reinforce with and without augmenting the state-space with accrued reward on two environments, Fishwood and Mini-Random.

We execute both MO Reinforce and MOCAC on this environment, with and without conditioning. The results are averaged over 10 runs and visible in Figure 3.3b.

As we can see, without augmenting the state-space, both algorithms plateau at a utility slightly above 10. This behavior can be explained as follows: even though the estimators are updated using the accrued rewards (so that the utility is applied on the episodic return), the agent has no way of knowing which of the random rewards it will receive at a new episode. As such, only action a_2 will guarantee a positive utility, regardless of the first random reward. On any other action, the agent would gamble on the $1/3$ probability that the random reward is optimal. Only choosing a_2 results in an average episodic utility of 10.67, corroborating the non-conditioned plots on Figure 3.3b.

In contrast, the conditioned variants are able to select the optimal action every time, as the accrued reward is included in the state. This results in a much higher utility.

Finally, we notice that, although the episode lasts only for 2 timesteps, MOCAC reaches the optimal behavior more quickly than MO Reinforce.

Fishwood In the second environment, a hiker sets up his camp near the river, and needs to prepare food for the evening. To do so, he needs both fish, which can be found in the river, as well as wood to make a fire, which can be found in the woods nearby. Two sticks of wood are needed to grill one fish, so in order to eat, the hiker needs both. This translates in a utility function:

$$u = \min\left(\texttt{fish}, \left\lfloor \frac{\texttt{wood}}{2} \right\rfloor\right).$$

Concretely, this environment has 2 states, the riverbank and the woods. Both moving out of, and staying in either state results in a probability of getting either a fish or a wood-stick. This process is repeated for a number of timesteps, until the episode ends.

We first perform experiments on Fishwood using the following parameters: $p_f = 0.25, p_w = 0.65, l = 13$ where p_f, p_w, l are the probabilities of getting fish, wood, and the duration of an episode, respectively. Results are shown in Figure 3.3a. We observe similar trends as with the previous experiments: the conditioned policies plateau at a higher utility than the non-conditioned ones, and MOCAC converges faster than MO Reinforce.

We believe a combination of two factors explain this result. Firstly, the agent is not guaranteed to receive either fish or wood, due to the stochasticity of these rewards. So, depending on the success rate of the executed action, the agent might need to retry it. Secondly, the episode duration is short, meaning the agent will not have time to compensate for ill-chosen actions nor bad luck, by a simple randomized process. Knowing what has already been gathered becomes important in the choice for future actions.

In general, we can conclude that augmenting the state-space with the accrued reward can be necessary to reach optimal performance. In both experiments this occurred due to stochasticity of the reward schemes. We therefore conclude that if the reward function is stochastic, conditioning on the accrued rewards is necessary to guarantee optimal performance.

Finally, we note that experiments performed on Split, Deep-Sea-Treasure and Minecart all use deterministic reward functions. In these cases, when we run the experiments with and without reward conditioning, we did not see any improvement in performance by conditioning the policies with the accrued reward (it even learns slightly slower since the state-space is bigger), there is thus no conditioning when these environments are used in Section 3.9.

3.7.3 Illustration on general monotonically increasing utility functions

Finally, we investigate the applicability of MOCAC with respect to different utility functions. Recall that in Equation 3.22 we only assume that the utility function u is monotonically increasing. Therefore, we expect MOCAC to also perform well on other utility functions beside the ones we have been using so far. MOCAC's critic uses a multivariate categorical distribution over the returns as an approximation for the true distribution. For each category, the utility function is applied on its corresponding $\boldsymbol{z}_{i\ldots k}$. This might lead to inaccuracies as $\boldsymbol{z}_{i\ldots k}$ covers a range of return values. Due to the non-linear nature of u this can potentially lead to inaccurate gradient updates for MOCAC's policy. Note that we can reduce these inaccuracies by increasing the number of categories C, at the expense of additional computation. To show the representational power of a categorical critic, we apply MOCAC with 5 utility functions with different properties on the Fishwood environment.

The utility functions used are the following:

$$
\begin{aligned}
u_1 &= r_0^2 + r_1^2 \\
u_2 &= r_0^2 + r_1 \\
u_3 &= \max(\lfloor r_0 \rfloor, \lfloor \frac{r_1}{2} \rfloor) \\
u_4 &= r_0 r_1 \\
u_5 &= \begin{cases} r_0 r_1 & \text{if } r_0 > 2 \\ 0 & \text{if } r_0 \leq 2, \end{cases}
\end{aligned}
$$

where r_0, r_1 is the return for the first (fish), second (wood) objective respectively. Since $p_f < p_w$, squaring r_0 in u_2 makes it more difficult to ascertain if the agent should focus on fish or wood. Focusing on fish results in a slightly higher utility as focusing on wood but, since the reward function is stochastic, the utility after each episode varies, often resulting in a lower utility compared to focusing on wood. For u_3, we focus on non-polynomial non-linearities by using $\max$ and $\lfloor . \rfloor$ operators. For u_4, both objectives are multiplied with each other. The non-linearity lies in the combination of both objectives, which contrasts with u_1 and u_2 that applied a non-linearity on each objective separately. Finally, u_5 introduces a threshold constraint.

We perform 20 runs for every utility function. We augment the state-space with accrued rewards as we have shown this is necessary to learn the optimal policy on Fishwood. Results are shown in Figure 3.4 for each utility function separately. The bottom right plot shows the normalized and aggregated performance across the 5 utility functions. We compare MOCAC with MO Reinforce. Both algorithms show a high standard deviation. This is because, due to the stochasticity of the reward function combined with the non-linearity of each utility function, the utilities received at the end of each episode vary greatly. Still, MOCAC systematically learns the optimal policy, receiving a normalized utility of 1 on average at the end of every episode. In contrast, MO Reinforce sometimes learns suboptimal policies even though it applies policy gradient on the episodic utility. We believe this occurs because of the high variance in utilities, even for similar action sequences compared to MOCAC which uses a baseline to reduce variance.

Thus, MOCAC can handle different types of utility functions and the variance introduced by their non-linearities. MO Reinforce, on the other hand, learns the optimal policy most of the time but not consistently, as on average it receives a normalized utility of 0.75.

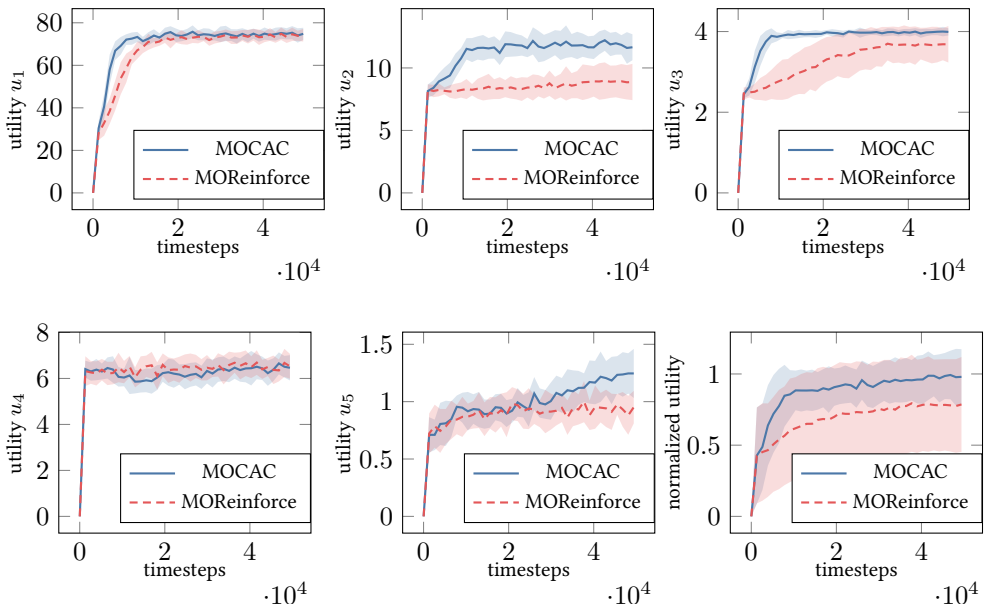


Figure 3.4: Results for diverse utility functions. The bottom right figure shows the average normalized performance over the 5 utility functions.

3.8 Benchmark environments

To test the effectiveness and sample-efficiency of MOCAC, we evaluate it on Deep Sea Treasure (Vamplew et al., 2011) and Minecart (Abels et al., 2019), two different MOMDP benchmarks from the MORL literature. We start by defining these benchmark environments.

3.8.1 Deep-Sea-Treasure

Deep-Sea-Treasure (DST) is a classic multi-objective benchmark (Vamplew et al., 2011) in which the agent controls a submarine in search for treasure. Deeper in the ocean lies higher-valued treasures, but it will take longer to reach them. Thus, there is a trade-off between treasure-value and time. We define r_0, r_1 as the rewards received for the treasure and time objectives, respectively.

DST is a grid-world environment, where the submarine moves on a 11×12, resulting in 132 different states. The state number is one-hot encoded, resulting in a 132-sized vector that is given as input to any of the actor and critic estimators.

We can arbitrarily choose the shape of the Pareto front by modifying the reward values provided by each treasure or by changing the layout of the ocean floor. For the experiments involving MOCAC, we have modified the original treasure values in such a way that every optimal `treasure` $\times$ `fuel` combination is evenly spread out on the convex coverage set. Treasure values are displayed on Figure 3.5. In the original environment, the treasures

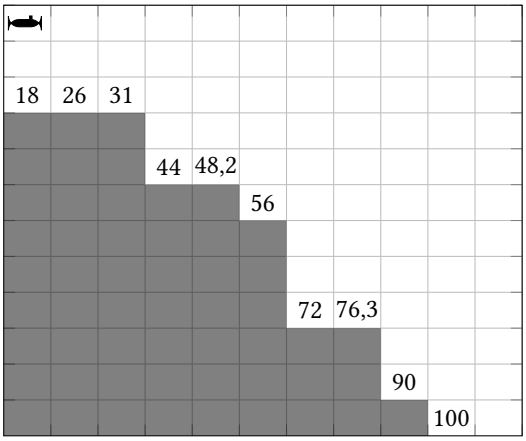


Figure 3.5: The *Deep Sea Treasure* environment with modified treasure compared to the original environment (Vamplew et al., 2011). Treasure values are made in such a way that every optimal $\texttt{treasure} \times \texttt{fuel}$ combination is evenly spread out on the convex coverage set. The agent starts in the top-left corner and tries to reach any of the treasures. Further treasures are worth more.

values at the bottom of the ocean are $1, 2, 3, 5, 8, 16, 24, 50, 74, 124$ respectively. Using 11 evenly-spread categories, as is done in our experiments, results in the first 5 treasures (half of the number of possible treasures) falling in the first category. A straightforward solution would be to increase the number of categories until all the treasures have their own category. However, due to the multivariate nature of the distribution, this results in an exponential increase of categories. Although this allows to accurately compute the estimated expected utility, it comes at a high computational cost as, at each critic-update, each category is updated according to all its neighbors. Thus, this results in a considerably higher wall time per update. Due to time-constraints, we have instead opted for a more evenly spread-out Pareto front. Another way to mitigate this problem is to use unevenly-spread categories. However, this assumes we know the spread of solutions of the Pareto front.

3.8.2 Minecart

Minecart (Abels et al., 2019) is a complex environment with a high-dimensional state-space. Starting at a base station, the agent controls a cart whose goal is to mine diverse ores from the mines scattered in the environment, and go back to the base to sell the ores. The agent moves a cart in a continuous 2-dimensional space. Internally, the state is represented by a 6-dimensional continuous vector containing the cart's velocity, position, angular rotation and cargo. We can either use this internal state or use the pixel frames of the environment's visual representation as observations. For the experiments involving MOCAC, we use the pixel representation. A frame of the environment can be seen in Figure 3.6.

The agent can execute 6 possible actions: it can accelerate, decelerate and rotate the cart

Figure 3.6: The *Minecart* environment. The base is located in the top-left corner, while the 5 mines are spread around the environment.

to the left or right. It can mine ores, which will only be effective if it is located in a mine. It can also simply do nothing.

Since each ore type represents a separate objective, we can vary the number of objectives by increasing the types of ores we can mine. Moreover, we can change the shape of the Pareto front by changing the number of mines present on the map, altering the mine locations, and changing the types and quantities of ores present in each mine. For the experiments involving MOCAC, there are a total of 2 objectives: the amount of ores mined, and the fuel consumption.

3.9 Experiments

We now test the effectiveness of MOCAC on both Deep-Sea-Treasure and Minecart. In all our experiments, we compare MOCAC with our proposed baseline algorithm, MO Reinforce, described in Section 3.4. Additionally, by using the MOMDP to MDP mappings described in Section 3.3, we use single-objective methods as a baseline. For this, we evaluate the well-known single-objective Advantage Actor-Critic (A2C) algorithm (Mnih et al., 2016) on these MDPs, as it is the one closest to our method. We call the baseline that executes A2C using the timestep, terminal mapping *A2C (timestep)*, *A2C (terminal)*, respectively. All experiments are averaged over 5 runs. The hyperparameters and neural network architecture used for each experiment are given in in Section A.2 of the Appendix. The code is publicly available online[2].

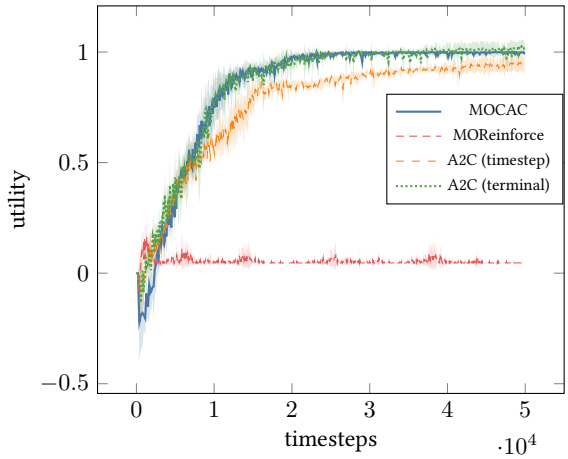


Figure 3.7: Results for Deep-Sea-Treasure, using linear utility with weight $w_0 = 0.9$. A2C (timestep) displays a slower convergence speed than MOCAC and A2C (terminal). As for most weights, is stuck at the first treasure.

3.9.1 Deep-Sea-Treasure

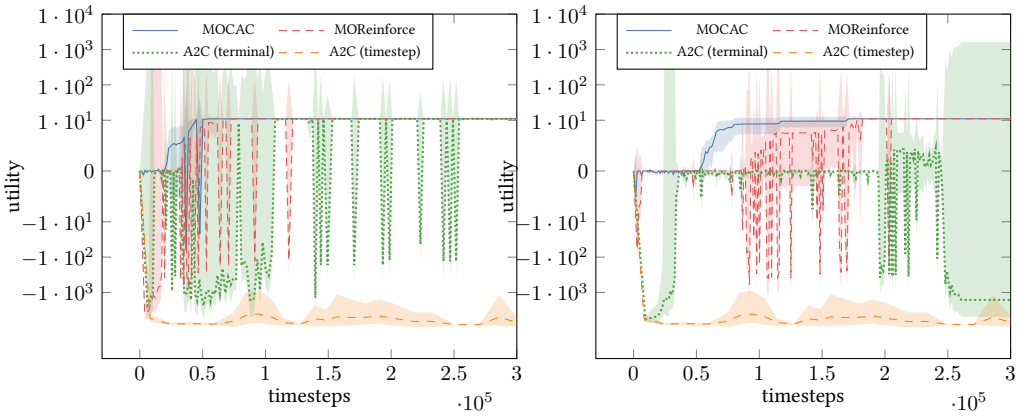
Linear Utility Function We consider two scenarios for this environment. Although our focus lies in the non-linearity of utility functions, we first show that MOCAC performs just as well on linear utility functions. In this scenario, we use a linearly weighted sum as utility function, i.e., $u = w_0 r_0 + (1 - w_0) r_1$ and $w_0 \in 0, 0.1, \ldots, 1$.

Non-linear Utility Function In a second scenario we consider a non-linear utility function since a key contribution of our proposed method is that it can cope with this class of functions. In this setting, a debt-ridden crew is seeking treasure to pay back their creditors before some deadline. If they are late, they have to pay a late-fee penalty, as well as interests for every additional timestep. We translate this scenario in the following utility function:

$$u = \begin{cases} \ln(1 + e^{(r_0 - d_0)}) & \text{if } r_1 \leq d_1 \\ \ln(1 + e^{(r_0 - d_0)}) - (r_1 - d_1)^2 - p & \text{if } r_1 > d_1, \end{cases} \tag{3.34}$$

where d_0 is the debt, d_1 the deadline, and p the penalty. The first term is a softplus function (Szandała, 2021), meaning that any treasure with a lower value than the debt will yield a zero reward, but the crew is of course free to keep any additional spoils. The other terms represent the interests and penalty. In this case, $d_0 = 45, d_1 = 10, p = 10$, resulting in the sixth treasure being optimal (out of ten).

[2]https://github.com/mathieu-reymond/mocac



(a) Accrued

(b) Not Accrued

Figure 3.8: Results for Deep-Sea-Treasure, using a non-linear utility function. Only the dedicated multi-objective approaches learn the optimal policy. A2C (terminal) shows unstable learning curves and A2C (timestep) fails to learn any decent policy.

Results

We first discuss the linear utility function. The results for $w_0 = 0.9$ can be seen on Figure 3.7. Results for the other weights are available in Figure A.1 of the Appendix. Regardless of the value for the weight w_0, MO Reinforce is not able to learn well because, unable to explore beyond the first treasure, it sticks to the easily obtainable reward the treasure provides. In contrast, both the A2C (timestep) and A2C (terminal) baselines learn the optimal policy. However, differences in learning speed appear depending on the weight value.

For example, low values for w_0 result in lower convergence speeds for A2C (terminal). Since it only receives non-zero rewards at the end of the episode, it spends more time exploring compared to A2C (timestep).

On the other hand, with high values for w_0, A2C (timestep)'s convergence speed decreases compared to the other algorithms, even though rewards are provided at every timestep. In this case, exploration is beneficial for A2C (terminal). It reaches later treasures earlier than A2C (timestep).

Regardless, MOCAC proves to be robust to the differences in weights, as it systematically learns the optimal policy. Moreover, its convergence speed is on par with or faster than the best performing baseline for each w_0.

With a non-linear utility function, the story becomes quite different, as can be seen in Figure 3.8.

As expected, A2C (timestep) is unable to learn any decent behavior. Since scalarization occurs at every timestep, u never receives the total time spent to find the treasure, meaning the deadline penalty is never applied. Because time is incorrectly taken into account, this baseline learns a policy that seeks the biggest treasure, but is also the one that is the furthest

away. This yields a poor episodic utility.

In contrast, since MOCAC, MO Reinforce and A2C (terminal) receive the correct utility at the end of each episode they are all able to learn the optimal policy. However, we do observe differences in learning speed and stability. For A2C (timestep), the harsh time constraints (the quadratic time factor as well as the penalty term) make exploration difficult. Moreover, since scalarization is applied, it occurs that high treasure values get negated by the time penalties, especially when the episode contains unnecessary steps. This results in a learner that requires more timesteps compared to the dedicated multi-objective approaches to find the optimal policy and often deviates from this policy after having discovered it.

In comparison, both MO Reinforce and MOCAC quickly reach the optimal treasure and do not collapse to suboptimal policies afterwards. Even though the time constraint penalizes the total utility, keeping track of the different objectives separately benefits the learning process. In terms of convergence speeds, MOCAC reaches the optimal treasure first but, as the episodes are short, the number of additional updates compared to the number of episodes amounts to an average of only 0.2%. Thus, it is not so much the sample efficiency but the reduction in variance – due to the advantage – that helps MOCAC perform better than MO Reinforce.

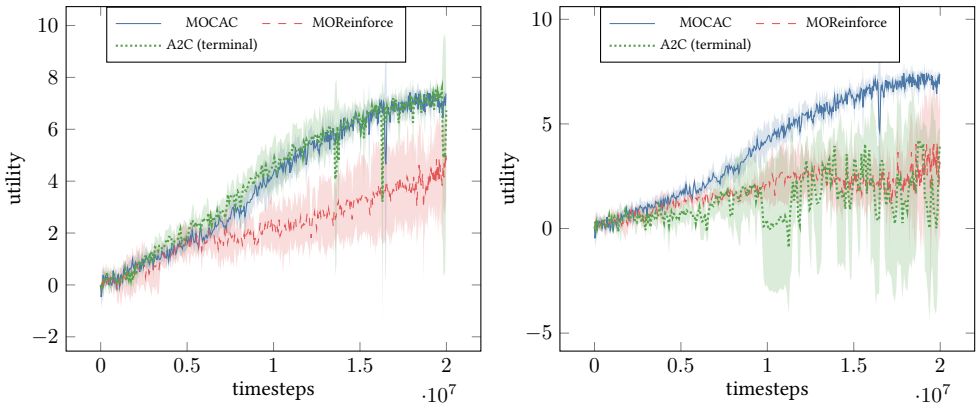
3.9.2 Minecart

For Minecart, we perform two sets of experiments. In the first, the agent is trained on the 6-dimensional, continuous state-space from the environment. In the second, the agent is trained on 84×84 pixel frames, using the same image pre-processing and convolutional network architecture as in Mnih et al. (2015).

Non-linear Utility Function As with the Deep-Sea-Treasure environment, we imagine a setting where a known non-linear utility function needs to be applied. The agent is a mining company with a contract to provide a specific amount of ores at an agreed-upon price. The leftover ores can be sold at market price. If there is a breach of contract due to insufficient amount of ores, a compensation penalty will be applied. Finally, fuel is seen as an additional expense. This can be formalised in the following utility function:

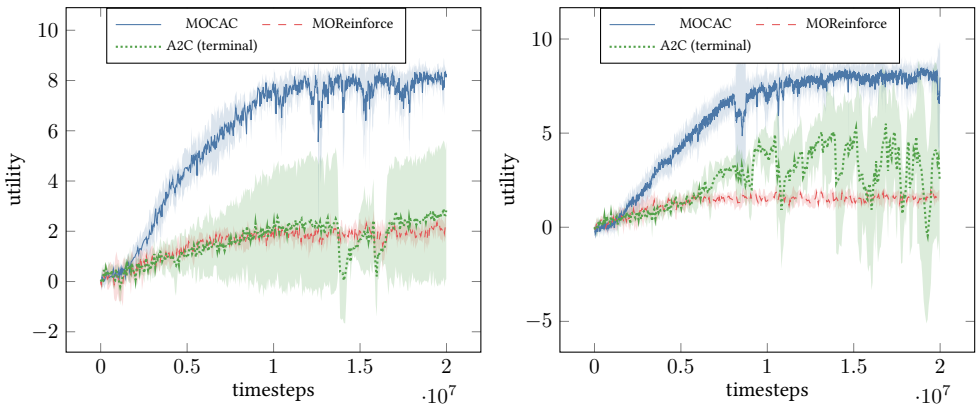
$$u = \begin{cases} t_0 p_0 + (r_0 - t_0)p_1 - r_1/20 & \texttt{if} \;\; t_0 \leq r_0 \\ r_0 p_0 - c - r_1/20 & \texttt{if} \;\; t_0 > r_0, \end{cases} \tag{3.35}$$

where t_0, p_0, p_1, c are the request ore amount, contract price, market price, and compensation penalty, respectively. In our scenario, $t_0 = 0.7, p_0 = 5, p_1 = 7, c = 2$.



(a) Accrued (b) Not Accrued

Figure 3.9: Results for Minecart using a non-linear u with a 6-dimensional continuous state-space. MOCAC outperforms all baselines. MO Reinforce performs on-par with A2C (terminal).



(a) Accrued (b) Not Accrued

Figure 3.10: Results for Minecart using a non-linear u with 84×84 pixel frames, similar to the Atari 57 suite. MOCAC outperforms all other algorithms.

Results

As can be seen in Figure 3.9 and Figure 3.10, for both sets of experiments MOCAC reaches the highest utility. The agent almost consistently fills its cart to capacity, and goes back to the base station to sell the minerals.

Looking at the baselines, we observe different behavior depending on the state-space used for training. For the 6-dimensional state-space, MO Reinforce on average performs on-

par with A2C (terminal), despite the higher variance and worse sample efficiency than its actor-critic counterpart. Both agents do not always fill their cart to full capacity, resulting in a lower utility.

With pixel frames their behavior is different. Although the network architecture is (except for the output) the same as MOCAC, some A2C (terminal) runs never learn to mine ores. MO Reinforce only partially fills its cart, usually just enough to reach the quota, but with nothing left to sell at market price.

All in all, combining the actor with a distributional critic in MOCAC is key to obtaining good utility.

3.10 Related work

Most of the recent work on MORL assumes an unknown, but linear, utility function. We discuss unknown utility functions in Chapter 5 and focus here on the setting where the utility function is decided upon beforehand. Most similar to our work is P. Zhang et al. (2021), which also learns a multivariate distribution over multi-objective returns. However, it does not consider accrued rewards, which we found necessarily for optimization with non-linear utility functions. Instead, the rewards are aligned with each other (e.g., separate rewards functions for killing monsters and for eating coins in Pacman), and serve as additional information for the policy, further confirming that taking into account different objectives is beneficial even if the utility function is known in advance.

Neil et al. (2018), Peng et al. (2018), and Tesauro et al. (2008) use linear utility functions, with the known issue that a small change in weights might lead to completely different policies (Vamplew et al., 2011). Van Seijen et al. (2017) learn separate Q-values per objective, optimized under the l_2 norm over the expected returns (and is thus related to SER). When selecting an action, the Q-values of the different objectives are summed up. Interestingly, Lin et al. (2020) take an opposite approach, by learning to decompose a single reward signal into different objectives.

Non-linear utility has been investigated in a tabular setting for SER by Van Moffaert et al. (2013b), who use a Chebyshev function. Moreover, monotonically increasing utility functions in general have been investigated in the (much simpler) bandit setting (Roijers et al., 2017), by modelling them using a Gaussian process and interacting with the user to obtain preference information. For MOMDPs, Hayes, Reymond, et al. (2021) propose a variant of Monte-Carlo Tree-Search (MCTS) to find the optimal policy for monotonically increasing utility functions. However, they require a model of the MOMDP.

A related field, called lexicographical RL, ranks the different objectives by preference (Gábor et al., 1998). For humans, this is less error-prone as putting absolute weights on objectives (Skalse et al., 2022). Like for the work discussed in this chapter, the preferences are known in advance. This gives rise to Lexicographic MDPs (K. Wray et al., 2015), a subset of MOMDPs, and Lexicographic POMDPs (K. H. Wray & Zilberstein, 2015). This has been used to optimize a primary reward function, while also optimizing auxiliary rewards (A. M. Turner et al., 2020), or avoiding negative side effects (Saisubramanian et al., 2021; A. Turner et al., 2020). Moreover, it has been used in the context of safe reinforcement learning, where the secondary objectives correspond to safety constraints (Skalse

et al., 2022), such as avoiding collisions and obeying to traffic rules in the autonomous driving domain (C. Li & Czarnecki, 2018). Instead of ranking the different objectives, we can require each objective to reach a minimal acceptable value (i.e., a threshold). This is called thresholded lexicographical ordering, and has been investigated in the tabular setting (Vamplew et al., 2011) by modifying tabular Q-learning, and then further extended for the multi-agent setting (Hayes et al., 2020).

The known-utility scenario also relates to constrained RL, where the policy optimizes a (single-objective) reward, subject to constraints. Often, these constraints are incorporated as a penalty signal into the original reward function, with manual weight selection (Levine & Koltun, 2013; Tamar & Mannor, 2013). However, since it is usually unknown how to weight the penalty signal, other works use a primal-dual algorithm to avoid manually setting these weights (Bhatnagar & Lakshmanan, 2012; Paternain et al., 2019; Tessler et al., 2019). These algorithms are less suited for a multi-objective approach, as they cannot arbitrarily combine the different objectives using a non-linear scalarization function.

3.11 Discussion

We proposed Multi-Objective Categorical Actor-Critic (MOCAC). To our knowledge, this is the first actor-critic RL algorithm that can handle MORL under the expected scalarized returns criterion, where the utility function can be non-linear. MOCAC takes into account accrued rewards and, in contrast to single-objective actor-critic RL algorithms, its critic only works if it learns a multivariate distribution over future returns, rather than an expected value over future returns.

In its current form, the multivariate distribution is represented as a categorical distribution, its main advantage being ease of computation. As a downside, the number of categories increases exponentially with the number of objectives. To scale to decision problems with many objectives, another representation is required, which we leave for future work. Another challenge is that the categorical distribution we use is not well-suited for decision problems for which the Pareto front contains solutions that are not evenly spread out. For example, in the original Deep Sea Treasure environment, the treasures values at the bottom of the ocean are $1, 2, 3, 5, 8, 16, 24, 50, 74, 124$ respectively. Using 11 categories, as was done in our experiments, results in the first 5 treasures (half of the number of possible treasures) falling in the first category. This is why, in our experiments we have modified the treasure values (see Section 3.8.1). While there are a number of ways to remedy this problem, such as increasing the number of categories, using categories of unequal sizes, or using a different number of categories per objective, this does not solve the more fundamental scalability issue encountered when using categorical distributions. For single-objective RL, quantile functions have been used to represent the distribution over returns, using a step-function with a fixed number of quantiles as approximation (Dabney et al., 2018; D. Yang et al., 2019). These quantiles can be learned, to best approximate the quantile function. We believe a similar approach could be used to further extend MOCAC. However, these extensions do not change the core idea behind MOCAC, which is combining distributional RL with accrued rewards to cope with non-linear utility functions, which is what we have focused on this chapter.

We show empirically that MOCAC can successfully learn in MOMDPs under ESR with a

known utility function. Furthermore, we show that it is much more sample-efficient and stable than all the proposed alternatives, clearly indicating that learning a distribution over the vectorial returns can convey important benefits in this class of problems.

Chapter 4

Improving our knowledge about the utility function

4.1 Introduction . . . 60
4.2 Multi-armed bandits . . . 61
4.3 Algorithms for best-arm identification . . . 62
4.3.1 Round robin . . . 62
4.3.2 Upper Confidence Bound . . . 62
4.3.3 Top-two Thompson Sampling . . . 63
4.4 Multi-objective multi-armed bandits . . . 65
4.4.1 Multi-objective top-two Thompson sampling . . . 67
4.4.2 Belief distribution of the utility function . . . 67
4.4.3 Comparison between Bayesian logistic regression and particle filtering . . . 69
4.4.4 Selecting queries for the decision maker . . . 71
4.4.5 Different query timings for multi-objective Top-two Thompson sampling . . . 72
4.4.6 Experiments . . . 72
4.5 POMCP for query optimization . . . 76
4.5.1 Simulating rollouts . . . 77
4.5.2 Transition model of MOMABs for simulated rollouts . . . 78
4.5.3 Aggregating belief-nodes together . . . 79
4.5.4 Evaluating rollouts . . . 80
4.5.5 Experiments . . . 81
4.6 Related work . . . 82
4.7 Discussion . . . 83

4.1 Introduction

Multi-objective optimization revolves around the decision maker's preferences. In Chapter 3, we demonstrated that using a dedicated multi-objective approach is beneficial even when we have full knowledge of the utility function. However, the chapter did not cover how to obtain this knowledge, and it was assumed to be available. Obtaining this knowledge can be challenging, as some computational processes may be too computationally expensive or demanding to be used as a reward function in the (MO)RL process. Additionally, some processes may require expensive machinery and specialized personnel to test in a laboratory setting. For example, a pharmaceutical company could design a new drug in silico and test its performance in vitro. In both cases, the utility function is expensive to evaluate, even though it gives an accurate utility score. In cases where the decision maker is a board of stakeholders, each proposition needs to be negotiated, adding another layer of complexity.

Although, in these cases, there is a cost of evaluating the utility function, each evaluation improves our understanding of the decision maker's preferences. The challenge, then, is to obtain as much information as possible over the decision maker's preferences with each additional evaluation. Thus, the proposed solution should be meaningful, since evaluating a Pareto-efficient policy provides more insight than evaluating a random policy. However, learning a Pareto-efficient policy is a challenging task, even when the utility is fully known. We argue that the two processes are intertwined, as understanding the preferences of the decision maker guides the RL learning process towards the optimal policy, but evaluating Pareto-efficient policies improves our understanding over these preferences.

Thus, we need to learn these two processes in conjunction. By evaluating solutions during the RL learning phase, we improve our knowledge over the utility function, which narrows the search-space of the optimal policy with respect to the user preferences. Thus, it becomes easier to refine and improve the solution over time. Moreover, this interactivity allows for our estimate of the utility function to be adapted to changing preferences or circumstances. As the decision maker's preferences evolve, the policy can be updated to reflect these changes. Finally, in case the decision maker is a person (or group of persons), their involvement in the learning process means they gain a better understanding of how it works and how their preferences are being taken into account. This can increase their trust in the system and make them more willing to use it. The question then is, how can we maximize our chances of learning the optimal policy, given the limited number of evaluations of the utility function? We address this question in this chapter, by optimizing the timing at which we evaluate the utility function, such that, given the current exploration of the search-space, we can propose a relevant solution to evaluate, that will maximize the improvement over our estimate of the utility function, such that this improvement will maximally improve subsequent exploration of the search-space.

To focus on the impact of the query timings, we study a specific class of MORL problems, called Multi-objective multi-armed bandits (MOMABs), which remove the sequential nature of the decision process. This simplifies the policy learning, for the profit of the query optimization.

We take a Bayesian approach, by learning a belief distribution over the preferences of the decision maker. Initially, this belief distribution will be highly uncertain about what these

preferences can be, sampling from it thus results in completely different utility function estimates. Our aim is to select actions that are optimal for any of the potential utility functions covered by the belief distribution. However, we include an additional action to our action-space: the query-action, that asks a query to the decision maker. Before selecting an action, we perform a number of simulations on a model of the environment. The best-performing action in the simulations is selected to be executed in the actual environment. The model used for simulations is also learned, and takes into account uncertainty about the environment's dynamics, again using a Bayesian approach. We call this algorithm *Multi-objective Partially Observable Monte-Carlo Planning* (MOPOMCP), and show that it significantly improves the chances of finding the optimal policy compared to interacting with the decision maker at fixed intervals.

4.2 Multi-armed bandits

In this work, we focus on a specific class of MORL problems, called Multi-objective multi-armed bandits (MOMABs). First, we introduce its single-objective counterpart, Multi-armed bandits (MABs) (Auer et al., 2002). The name comes from slot machines, also called one-armed bandits, found at the casino. When the gambler pulls the lever (or arm) of a slot machine, its reels spin, sometimes resulting in a winning combination that pays out a certain amount of money. Assuming each slot machine of the casino is configured differently, our gambler should try to figure out which machine has the highest chance of returning a winning combination of reels. In RL terms, one can see a MAB as a single-state, single-step RL problem, where each action corresponds to pulling the lever of a different slot machine (in the bandit literature, this is referred to as *pulling an arm*). The reward function is stochastic, as each arm has a different probability of returning a winning payout. Since this is a single-step problem, the episode ends after pulling one of the arms. The Q-value of each arm is thus the average reward obtained by this arm, which, in the limit, is equal to the mean of the reward distribution. Analogously as the definition of the optimal policy in Section 2.3, the optimal policy amounts to choosing the action with the highest Q-value.

Due to the uncertainty associated with the reward of each policy execution (or arm-pull), it is challenging to learn the optimal action as quickly as possible. As such, MABs are well-suited to study the exploration-exploitation trade-off, as one needs to explore different actions to better understand their associated reward-distributions (and thus gain confidence in the estimated Q-values), but one does not want to try out too many alternatives, as they are costly and suboptimal.

MABs provide a theoretical framework for many real-world problems. Examples include online advertisement, where we want to select the advert that has the highest probability of being clicked on by a user (Chapelle & Li, 2011; Rhuggenaath et al., 2019), the mitigation of epidemics (P. Libin et al., 2019) and wind farm control, where we want to select the orientation of the blades that maximize the power output (Bargiacchi et al., 2018).

For our problem setting, the multi-objective variant of this framework allows us to study the timing of the utility function evaluation, without having to take into account the sequential learning part of the policy. Instead, we have a fixed number of possible policies, where the optimal policy can change depending on the utility function. Learning the Q-values and associated optimal policy is focused on single-step executions instead of multi-

step executions, but gives us insight on the convergence speed and probability of learning the optimal policy within the allotted training steps.

Formally, we define a MAB as a set of parametric reward distributions $\mathcal{P}_{\theta_0}, \ldots, \mathcal{P}_{\theta_A}$ with parameters $\theta_0, \ldots, \theta_A$ respectively, where executing action $a \in \mathcal{A}$ returns a sample $r \sim \mathcal{P}_{\theta_a}$:

$$\mathfrak{B} = \{\mathcal{P}_{\theta_0}, \ldots, \mathcal{P}_{\theta_A}\}. \tag{4.1}$$

For example, assuming the rewards are normally distributed, θ_a is characterized by the mean μ_a and standard deviation σ_a of the normal distribution, resulting in $\mathfrak{B} = \{\mathcal{N}(\mu_0, \sigma_0), \ldots, \mathcal{N}(\mu_A, \sigma_A)\}$.

The optimal policy, or arm, is defined as:

$$\pi^* = \arg\max_{a \in \mathcal{A}} Q(a) = \arg\max_{a \in \mathcal{A}} \mu_a, \tag{4.2}$$

where μ_a is the mean of the distribution $\mathcal{P}_{\theta_a}$.

Our problem consists in maximizing our chances of finding the optimal policy, given a limited budget of utility function evaluations and of policy evaluations. This corresponds, in the MAB literature, to the best-arm identification setting (Audibert et al., 2010).

4.3 Algorithms for best-arm identification

Our goal is to maximize our chances of finding the optimal policy, given a fixed budget. In this Section, we introduce some of the most well-known single-objective algorithms for this setting.

4.3.1 Round robin

A naive approach would be to select each action the same number of times, until the budget is spent, and then selecting the action with the highest Q-value estimate. This is called the *round-robin* strategy. The downside of this approach is that valuable budget is spent on arms for which we are increasingly certain that they are suboptimal (i.e., the estimated Q-values are much lower than for other arms). Instead, this budget could be spent on improving our confidence in the top arms (i.e., the arms with similar Q-values). Since we do not exploit any of our current knowledge (i.e., estimated Q-values), round-robin is a pure exploration strategy. The algorithm is shown in Algorithm 3.

4.3.2 Upper Confidence Bound

Instead of pure exploration, one would like to also take advantage of the knowledge gained in previous interactions. One of the most famous algorithms to tackle MABs is the Upper Confidence Bound (UCB) algorithm, which balances exploration and exploitation using a frequentist approach (Auer et al., 2002). UCB is based on the idea that actions that have been selected many times are unlikely to see significant changes in their Q-values. Instead,

Algorithm 3 Round-robin

Require: T budget
1: $p, Q \leftarrow 0$, initial pull count, Q-value for each arm to 0
2: **for** $t \in [T]$ **do**
3: $a \leftarrow t \mod |\mathcal{A}|$, select next arm
4: $r_a \sim \mathcal{P}_{\theta_a}$, pull a
5: $p_a \leftarrow p_a + 1$, increment pull count
6: $Q(a) \leftarrow \frac{1}{p}(Q(a)(p_a - 1) + r_a)$, update estimated mean
7: **end for**
8: **return** $\arg\max_{a \in \mathcal{A}} Q(a)$

Algorithm 4 Upper Confidence Bound

Require: T budget
1: $p, Q \leftarrow 0$, initial pull count, Q-value for each arm to 0
2: **for** $t \in [T]$ **do**
3: $a \leftarrow \arg\max_a Q(a) + \beta\sqrt{\frac{\ln t}{p_a}}$, select next arm based on UCB
4: $r_a \sim \mathcal{P}_{\theta_a}$, pull a
5: $p_a \leftarrow p_a + 1$, increment pull count
6: $Q_a \leftarrow \frac{1}{p}(Q_a(p_a - 1) + r_a)$, update estimated mean
7: **end for**
8: **return** $\arg\max_{a \in \mathcal{A}} Q_a$

one should select actions with a high potential for reward: actions that have been selected few enough times that, given their current Q-values, they can still become optimal. Concretely, at time t, the arm is selected as follows:

$$a_t = \arg\max_a Q_{t-1}(a) + \beta\sqrt{\frac{\ln t}{N_{t-1}(a)}}, \tag{4.3}$$

where $N_{t-1}(a)$ is defined as the number of times action a has been selected at time $t-1$, and β a constant weighting the importance of the exploration factor. We see that the exploration term decreases with $N_{t-1}(a)$, thus preferring promising arms that have been selecting less often. The downside of this algorithm, next to the sensitivity of the hyperparameter β, is its inability to incorporate prior knowledge. The algorithm is shown in Algorithm 4.

4.3.3 Top-two Thompson Sampling

In contrast to the frequentist approach taken by UCB, we can use a Bayesian approach, which naturally incorporates prior knowledge about data and statistics (Thompson, 1933). A well-known Bayesian algorithm for best arm identification is called Top-two Thompson Sampling (TTTS) (Russo, 2016). The primary goal of TTTS is to distinguish the best arm from the second-best arm. The stronger this distinction, the highest confidence it has that the estimated best arm is indeed the optimal arm. Since the other arms are worse than the

Algorithm 5 Best arm sampling

Require: $\mathcal{H}$ history of pulls, ϕ prior parameters
1: **for** $a \in \mathcal{A}$ **do** ▷ for each arm, sample a mean
2: sample $\hat{\theta}_a \sim \mathcal{P}(\theta_a \mid \mathcal{H}_a)$
3: $\hat{\mu}_a \leftarrow \hat{\theta}_a$ ▷ Mean based on sampled parameters
4: **end for**
5: **return** $\arg\max \hat{\mu}_a$ ▷ arm of the highest estimated mean

second-best arm, their ordering does not matter, so we should avoid spending our budget on them.

To distinguish arms, TTTS maintains a belief distribution over each reward distribution $\mathcal{P}_{\theta_a}, a \in \mathcal{A}$. That is, TTTS estimates the parameters θ_a based on the history of observed samples $\mathcal{H}_{a,t} = \{r_0, \ldots, r_{t-1}\}$ from $\mathcal{P}_{\theta_a}$.

We would like to compute the probability distribution $\mathcal{P}(\theta_a \mid \mathcal{H}_{a,t})$ over the possible distribution parameters, given the history $\mathcal{H}_{a,t}$. This is called the *posterior distribution*. Initially, when our history $\mathcal{H}_{a,t}$ is empty, we are uncertain about θ_a and use default parameters ϕ_a. The distribution $\mathcal{P}(\theta_a \mid \phi_a)$ is called the *prior distribution*. According to Bayes' theorem (Bishop & Nasrabadi, 2006), we have:

$$\begin{aligned} \mathcal{P}(\theta_a \mid \mathcal{H}_{a,t}) &= \frac{\mathcal{P}(\mathcal{H}_{a,t} \mid \theta_a)\mathcal{P}(\theta_a \mid \phi_a)}{\mathcal{P}(\mathcal{H}_{a,t})}, \\ \mathcal{P}(\theta_a \mid \mathcal{H}_{a,t}) &\propto \mathcal{P}(\mathcal{H}_{a,t} \mid \theta_a)\mathcal{P}(\theta_a \mid \phi_a). \end{aligned} \tag{4.4}$$

We ignore $\mathcal{P}(\mathcal{H}_{a,t})$ since it does not depend on θ_a.

At time t, TTTS samples from a Bernoulli distribution (typically with $p = 0.5$ of success) $b \sim \mathcal{B}(p)$ to decide if it should pull the best arm. The ordering of arms is decided by sampling from each of the belief distributions, and sorting the arms according to their associated sample (see Algorithm 5). When b is a success, we pull the best arm. Otherwise, we aim to pull the second-best arm. The second-best arm is decided by saving the sampled best arm, and then resampling from each belief distribution until the resampled best arm is different from the saved best arm. That arm is then pulled.

At $t = 0$, when our beliefs have no information about the reward distributions, each arm is equally likely to be selected as best. However, as we pull arms, our belief distributions become increasingly informative, and the likelihood of the highest ranked arm corresponding to the optimal arm increases as well.

We repeat this process until the budget has been exhausted. At this point, the arm associated with the belief distribution with the highest mean is identified as the best arm. We show the algorithm in Algorithm 6.

Algorithm 6 Top-two Thompson sampling

Require: T budget, ϕ prior parameters, p success probability
1: $\mathcal{H} \leftarrow 0$, empty history for each arm
2: **for** $t \in [T]$ **do**
3: $i \leftarrow$ Algorithm 5 $(\mathcal{H}, \phi)$ ▷ arm of the highest estimated mean
4: sample $b \sim \mathcal{B}(p)$
5: **if** $b = 1$ **then**
6: $\mathcal{H}_i \leftarrow \mathcal{H}_i \cup r_t, \quad r_t \sim \mathcal{P}_{\theta_i}$ ▷ pull arm and add to history of arm-pulls
7: **else** ▷ sample second-best arm
8: **repeat**
9: $j \leftarrow$ Algorithm 5 $(\mathcal{H}, \phi)$
10: **until** $i \neq j$
11: $\mathcal{H}_j \leftarrow \mathcal{H}_j \cup r_t, \quad r_t \sim \mathcal{P}_{\theta_j}$ ▷ pull arm and add to history of arm-pulls
12: **end if**
13: **end for**
14: **return** $\arg\max_{a \in \mathcal{A}} \bar{\mu}_a$ ▷ arm with the highest sample mean

4.4 Multi-objective multi-armed bandits

Similarly as for MORL, in case we have vectorial rewards and the utility function is unknown, we cannot rank the rewards, and thus cannot straightforwardly use either UCB or TTTS. In this section, we formalize MABs with multiple objectives, and how they incorporate the utility function.

Formally, a multi-objective multi-armed bandit (MOMAB) (Drugan & Nowe, 2013) is a tuple $\mathfrak{B} = \langle \boldsymbol{\mathcal{R}}, u \rangle$, where $\boldsymbol{\mathcal{R}}$ is a set of parametric multivariate stochastic reward functions $\boldsymbol{\mathcal{P}}_{\theta_0}, \ldots, \boldsymbol{\mathcal{P}}_{\theta_A}$, and u is the utility function defining the preferences of the decision maker. For MOMABs, each $\boldsymbol{\mathcal{P}}_{\theta_a}$ is a multivariate distribution with the number of dimensions equal to the number of objectives.

$$\boldsymbol{\mathcal{R}} = \{\boldsymbol{\mathcal{P}}_{\theta_0}, \ldots, \boldsymbol{\mathcal{P}}_{\theta_A}\}. \tag{4.5}$$

The optimal policy, or optimal arm, is the arm resulting in the maximal utility. Like in MORL, this depends on the optimization criterion, which is either ESR or SER (as defined in Section 2.5).

For SER, the optimal policy corresponds to the arm with the highest utility with respect to its Q-value. This is equivalent to applying the utility function on the mean of the reward distribution:

$$\pi^* = \arg\max_{a \in \mathcal{A}} u(\boldsymbol{Q}(a)) = \arg\max_{a \in \mathcal{A}} u(\boldsymbol{\mu}_a), \tag{4.6}$$

where $\boldsymbol{\mu}_a$ is the mean of the multivariate distribution $\boldsymbol{\mathcal{P}}_{\theta_a}$.

For ESR, the optimal policy corresponds to the arm with the highest expected utility over rewards. This means we need to weight the sample with their probability of being sampled:

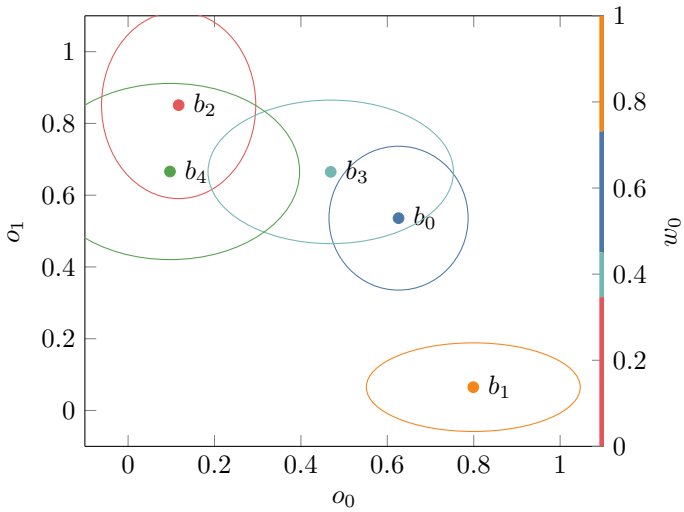


Figure 4.1: Example of a 5-arm, 2-objective MOMAB, where the reward distribution for each arm is represented by a multivariate normal distribution. The colored ellipse is drawn at 2 standard deviations from the mean. As arm b_4 is dominated by the other arms, it cannot be the optimal arm. Since the non-dominated coverage set (i.e., the other arms) is convex, each of its arms can be optimal for a linear utility function u, depending on its weight. The right y-axis represents the 1-dimensional simplex, from which u's weight can be sampled. This simplex is split in colored sections, where the color of each section corresponds to the weight-values for which the same-colored arm is optimal.

$$\pi^* = \arg\max_{a \in \mathcal{A}} \int_{-\infty}^{\infty} u(x) \mathrm{PDF}_a(x) dx, \tag{4.7}$$

where $\mathrm{PDF}_a(x)$ is the probability density function of $\boldsymbol{\mathcal{P}}_{\theta_a}$. Analogously to Chapter 3, this means that estimating the mean of the reward distribution is insufficient to learn the optimal policy, one needs to estimate the whole distribution. We believe this might pose additional challenges and is an interacting avenue for future work (see Section 7.1), as the ESR setting has been understudied compared to SER.

Figure 4.1 shows an example of a 2-objective MOMAB, with 5 arms. Each ellipse represents the multivariate normal distribution (up to 2 standard deviations from the mean) associated with its arm. We see that 4 of the 5 arms are non-dominated and can potentially be the optimal arm under SER, depending on the utility function. The right y-axis shows the 1-dimensional simplex associated with the weights of linear utility functions. Weights on the axis are colored according to their optimal arm.

4.4.1 Multi-objective top-two Thompson sampling

For single-objective optimization, TTTS is an efficient algorithm for best-arm identification. We propose to extend it to the multi-objective setting, by learning multivariate belief distributions over the arms. We call this algorithm Multi-objective top-two Thompson sampling (MOTTTS).

An important aspect of MOMABs is the inclusion of the utility function u. This function is initially unknown. To find the best arm, we need to have an understanding of u. Similarly as for arms, we keep a belief distribution over u, which we improve over time by interacting with the decision maker. Using this belief distribution, we can sample utility function estimates $\hat{u}$ and rank the multivariate samples coming from the arm belief distributions. This allows us to pull arms following the same strategy as the TTTS algorithm. Next, we explain how to represent this belief distribution, and how it changes when adding new datapoints.

4.4.2 Belief distribution of the utility function

While the utility function could be a formal process returning an absolute score, in many real-world problems, the decision maker is human. Humans find it challenging to express their preferences in absolute terms (e.g., "I like this movie 0.4 much"), as using numbers to express preferences can be unnatural and prone to errors (Tesauro, 1988). Additionally, values may change (Sidney, 1957) depending on the user's mood, which can be influenced by seemingly trivial factors like the weather (Forgas, 1995; Sirakaya et al., 2004). On the other hand, expressing preferences in relative terms (e.g., "I prefer movie A over B") is easier for humans and tends to be more consistent over time (Tesauro, 1988; Zoghi et al., 2014). Therefore, we focus on representing the utility function using relative feedback. We can translate this process as a binary classification task where, given two propositions, the goal is to predict if the first proposition is preferred over the second one. Formally, given two propositions $\boldsymbol{r}^0, \boldsymbol{r}^1$, we define $\succ : \mathbb{R}^n \times \mathbb{R}^n \to \{1, 0\}$ as the binary preference operator, where $\boldsymbol{r}^0 \succ \boldsymbol{r}^1$ outputs 1 when the decision maker prefers $\boldsymbol{r}^0$ over $\boldsymbol{r}^1$, and 0 otherwise. Thus, we define each interaction with the decision maker as a pair, where the first element is the two propositions to compare, and the second element is the decision maker's answer:

$$\langle \langle \boldsymbol{r}^0, \boldsymbol{r}^1 \rangle, \boldsymbol{r}^0 \succ \boldsymbol{r}^1 \rangle. \tag{4.8}$$

We keep each interaction in an interaction history $\mathcal{H}^q$, which is used to update the belief distribution over u.

In Interactive Thompson sampling (ITS) (Roijers et al., 2017), an algorithm for regret minimization for MOMABs, the authors propose to use Bayesian logistic regression (Bishop & Nasrabadi, 2006) for linear utility functions to predict the decision maker's relative preferences. This approach enables them to maintain a belief over the weights of the linear utility function, which are drawn from the $n-1$ simplex $\mathbb{S}^{n-1}$. However, the posterior distribution computed using exact Bayesian logistic regression does not have a closed form solution. Instead, one option is to estimate the posterior distribution using the Laplace approximation, which approximates the true posterior distribution to a multivariate normal

distribution (Bishop & Nasrabadi, 2006). The drawback is that the sampled $\boldsymbol{w} \sim \mathcal{N}$ do not adhere to the simplex constraints, and thus need to be normalized. This has the downside of further degrading the original approximation. Another downside is that the Laplace approximation uses gradient descent to fit the posterior distribution to the datapoints, which is a more expensive computational process compared to the closed-form posterior beliefs of the reward distributions.

Due to these downsides, we propose to use particle filtering (Doucet et al., 2000; Gordon et al., 1993) as a belief distribution over the utility function. Particle filtering uses a set of particles to represent a distribution, where each particle is weighted according their likelihood given the datapoints. Sampling from this belief distribution then amounts to sampling a particle according to their likelihood. Formally, given a set of particles $x_0, \ldots, x_P$, where P is the total number of particles, each x_i is associated with a particle weight $\boldsymbol{\omega}_i$, whose value is:

$$\boldsymbol{\omega}_i = \mathcal{P}(x_i \mid \mathcal{H}^q). \tag{4.9}$$

The mean of the particle distribution is then the weighted average of the particles:

$$\boldsymbol{\mu} = \sum_{i=0}^{P} \boldsymbol{\omega}_i x_i. \tag{4.10}$$

Since we can freely choose how we define our particles, one advantage of particle filtering is that it can approximate distributions of arbitrary shapes. When the utility function is linear, as is the case in ITS, we can represent u by its weights (see Equation 2.5). Since the weights from u belong to $\mathbb{S}^{n-1}$ (Equation 2.6), we sample our particles from it. Thus, contrary to the Bayesian logistic regression used in ITS, our belief distribution using particle filtering adheres to the simplex constraints. However, as a downside, the number of particles required to accurately represent the $n-1$ simplex increases exponentially with the number of objectives. One way to alleviate this problem is to use a pruning and resampling strategy, where particles with a likelihood below a certain threshold are pruned and replaced with new, more likely particles (Hol et al., 2006). We leave the many-objectives setting for future work, and focus on MOMABs with a limited number of objectives.

At the beginning of training, we assume that we do not possess any knowledge over u. Thus, initially all particles are weighted equally. However, with each new datapoint, the likelihood of each particle is updated. Intuitively, we give a high likelihood to all particles that match the decision maker's answers to past queries, and a low likelihood to the other particles. Given our history $\mathcal{H}^q$ of relative queries, the weight $\boldsymbol{\omega}$ of particle x is defined as:

$$\boldsymbol{\omega} = \prod_{h \in \mathcal{H}^q} |(\boldsymbol{r}_h^0 \succ \boldsymbol{r}_h^1) - \eta|(\boldsymbol{\omega}^\top \boldsymbol{r}_h^0 \geq \boldsymbol{\omega}^\top \boldsymbol{r}_h^1), \tag{4.11}$$

where η accounts for potential mistakes, or change of preference from the decision maker. Thus, when $\eta = 0$, only the particles of weights for which *all* answers of the decision maker correspond to the solution with the highest utility have a non-zero probability of being sampled. Moreover, these particles are equally likely.

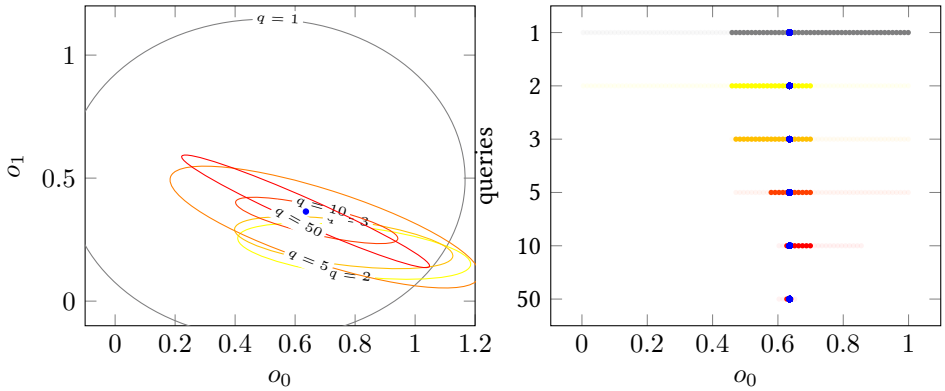


Figure 4.2: A side-by-side comparison of both types of belief distributions. In both cases, we show the evolution of the belief distribution with the size of the history of relative queries (1,2,3,5,10,50). The same queries were used for each belief distribution. On the left, we show the belief distribution using Bayesian logistic regression. As the size of the history increases, the covariance matrix becomes smaller. We also note that the weights become correlated, as they should sum to 1. Despite the decrease in variance, we see that, even after 50 queries, significantly different weights can be sampled. On the right, we show the belief distribution based on particle filtering. We see that the number of potential particles quickly becomes concentrated around the true mean.

Figure 4.2 shows a side-by-side comparison of both types of belief distributions, with Bayesian logistic regression, particle filtering to the left, right, respectively of the plot. We show the evolution of the belief distribution as more queries are added to the interaction history, and the unknown utility function's weight (in blue). Although, after 50 queries, the Bayesian logistic regression belief is centered around the true mean, and learns that both weights are correlated, the standard deviation is high, and significantly different weights can be sampled. In contrast, the only particles with a non-zero likelihood are the particles close to the true mean. Moreover, we observe that, with just a few queries, the number of potential particles has been greatly reduced.

4.4.3 Comparison between Bayesian logistic regression and particle filtering

By modifying ITS (Roijers et al., 2017), the algorithm that introduced Bayesian logistic regression for linear utility functions with relative preferences, and replacing its belief with particle filtering, we can compare the performance of both approaches.

ITS aims to minimize the cumulative regret. Since in regret minimization there is no budget limit, as the regret should be minimal for any budget, the setting differs from our best arm identification setting. Nonetheless, keeping an accurate belief over the utility function is crucial in the regret minimization setting as well, since an inaccurate belief increases the chances of pulling a suboptimal arm, which increases the regret.

At each timestep, ITS samples twice from its arms and utility belief distributions. It then

Algorithm 7 Interactive Thompson Sampling with Particle filtering

Require: ϕ prior parameters on reward distribution, prior parameters on utility weight distribution

1: $\mathcal{H}^A \leftarrow 0$, empty history for each arm
2: $\mathcal{H}^q \leftarrow 0$, empty interaction history
3: **while** 1 **do**
4: x_x, x_y with $x, y \sim \mathcal{P}(\boldsymbol{\omega}_{0:P} \mid \mathcal{H}^q)$ ▷ 2 samples from belief distribution over u
5: $i \leftarrow$ Algorithm 5 $(\mathcal{H}, \phi, x_x)$ ▷ arm of the highest estimated mean w.r.t. x_x
6: $j \leftarrow$ Algorithm 5 $(\mathcal{H}, \phi, x_y)$ ▷ arm of the highest estimated mean w.r.t. x_y
7: $\mathcal{H}_i^A \leftarrow \mathcal{H}_i^A \cup \boldsymbol{r}_t, \boldsymbol{r}_t \sim \boldsymbol{\mathcal{P}}_{\theta_i}$ ▷ pull first sampled arm and add to history of arm-pulls
8: **if** $i \neq j$ **then**
9: $\mathcal{H}^q \leftarrow \mathcal{H}^q \cup \langle\langle \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\mu}}_j \rangle, \hat{\boldsymbol{\mu}}_i \succ \hat{\boldsymbol{\mu}}_j \rangle$ ▷ Query user on current mean estimates and add to interaction history
10: **end if**
11: **end while**

ranks its arms based on the sampled utility. Since the beliefs where sampled twice, there are two arms a_i, a_j ranked first. ITS always pulls a_i, the top arm from the first set of samples. However, ITS assumes that, when a_i and a_j are different, it might be due to uncertainty over the utility function, and we should thus query the decision maker. The queries should ensure those two top arms will be ranked differently in a subsequent timestep (i.e., one arm will be better than the other). Thus, ITS proposes to use the estimated means $\hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\mu}}_j$ of a_i, a_j, respectively as solutions to be ranked by the decision maker. The interaction $\langle\langle \hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\mu}}_j \rangle, \hat{\boldsymbol{\mu}}_i \succ \hat{\boldsymbol{\mu}}_j \rangle$ is then added to the interaction history.

We compare the difference in performance of the two types of belief distributions by executing ITS on 10000 randomly generated 2-objective MOMABs, with randomly generated utility functions. For each MOMAB, we make 100000 experiments. For each experiment, we execute ITS for 2000 timesteps, as is done in the original paper (Roijers et al., 2017). All hyperparameters are the same as in the original experiments, and we use 100 particles equally-spaced over the $n-1$ simplex, initialized with uniform probability-weights. We plot the cumulative regret over time, averaged over all experiments, in Figure 4.3.

Figure 4.3 shows that, on average, using particle filtering leads to a lower cumulative regret than using Bayesian logistic regression as a belief distribution for u. Since the generated MOMABs are different, it is more difficult to find the optimal arm for some than for others. This means the cumulative regret inherently varies across MOMABs. As such, we did not find any insight in including the standard deviation in the plot. Instead, we computed the number of MOMABs for which using particle filtering results in a lower final regret. This is the case for 62% of the MOMABs, indicating some variability in performance using particle filtering. But, as it outperforms Bayesian logistic regression on the majority of the MOMABs, this confirms that, using a representation that inherently adheres to the simplex constraints is beneficial for MOMABs with linear utility functions. Moreover, ITS uses the estimated means $\hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\mu}}_j$ to query the decision maker. Over time, these values are unlikely to change significantly. Thus, over time, the decision maker receives similar queries. While

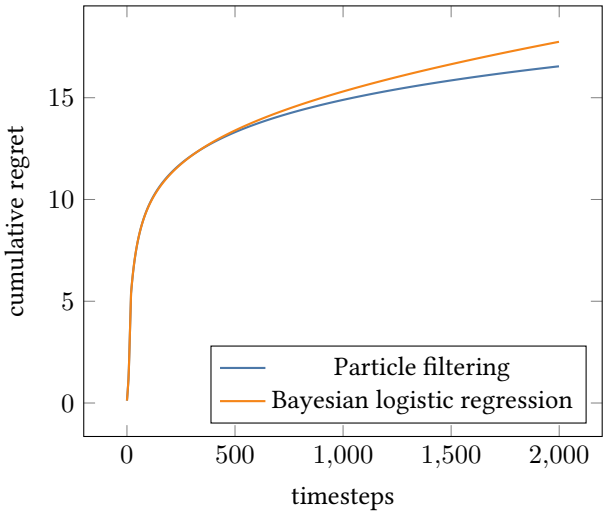


Figure 4.3: Cumulative regret of ITS using both types of belief distributions, averaged over 10000 MOMABs, with 100000 experiments per MOMAB. Since these MOMABs are randomly generated, and thus present their own peculiarities, we did not find any insight in including the standard deviation in the plot. Instead, we computed the number of MOMABs for which using particle filtering results in a lower regret, and determined this to be the case for 62% of the MOMABs.

having duplicate queries (or almost-duplicate queries) can be beneficial for Bayesian logistic regression, as it reduces the estimated standard deviation of the multivariate normal (see Figure 4.2), it is less beneficial for particle filtering, as these duplicate queries do not change the boundaries between successful and unsuccessful particles. Thus, this provides an opportunity to ask fewer questions to the decision maker. This means that, next to the type of belief distribution to consider for u, another aspect to take into consideration is which types of queries to ask the decision maker.

4.4.4 Selecting queries for the decision maker

Although pairwise comparisons are more reliable in terms of human answers, they provide less information than absolute scores, and are thus less effective to estimate the utility function. It is thus important that the pairwise comparisons are informative and realistic. For the comparisons to be realistic, we select the solution pairs based on our current belief distributions over arms. Moreover, since our goal is to distinguish the top-two arms, we aim to provide queries that further discriminate the top-two arms in terms of utility.

Inspired from ITS, we base our query-selection mechanism on Thompson sampling (Thompson, 1933). However, unlike ITS, which select queries based on the estimated means over arms, which results in potential similar queries over time, we base our queries on the samples themselves. This allows to increase variety in queries, while still focusing on discriminating the top arms.

Algorithm 8 Query selection

Require: $\mathcal{H}^A$ pull history for each arm, $\mathcal{H}^q$ interaction history

1: $\hat{u} \sim \mathcal{P}(\cdot \mid \mathcal{H}^q)$ ▷ sample utility function based on utility-belief
2: $\boldsymbol{r} \sim \boldsymbol{\mathcal{P}}_{\hat{\theta}_a} \quad \forall a \in \mathcal{A}$ where $\hat{\theta}_a \sim \mathcal{P}(\cdot \mid \mathcal{H}_a^A)$ ▷ sample each arm-belief
3: $k \leftarrow \arg\max_{a \in \mathcal{A}} \hat{u}(\boldsymbol{r}_a)$ ▷ top sample w.r.t. sampled utility function
4: $l \leftarrow \arg\max_{a \in \mathcal{A} \setminus \{k\}} \hat{u}(\boldsymbol{r}_a)$ ▷ second-top sample w.r.t. sampled utility function
5: **return** $\langle \langle \boldsymbol{r}_k, \boldsymbol{r}_l \rangle, \boldsymbol{r}_k \succ \boldsymbol{r}_l \rangle$ ▷ return interaction with decision maker

First, we sample a utility function estimate $\hat{u}$ from our belief distribution. Next, for each belief distribution over arms, we sample a vectorial reward $\hat{\boldsymbol{r}}_a \sim \boldsymbol{\mathcal{P}}_{\theta_a}$. For each arm a, we compute its utility $\hat{u}(\hat{\boldsymbol{r}}_a)$ using the corresponding sampled rewards and the sampled weights. We can then rank the arms according to their computed utility. We give the samples corresponding to the top-two ranked arms as a query to the decision maker. This allows us to have varied queries (as they are based on samples), that focus on a narrow region of the utility-space (the region that distinguishes the first and the second arm).

4.4.5 Different query timings for multi-objective Top-two Thompson sampling

Using particle filtering as belief distribution representation for the utility function, and using a Thompson sampling based approach to select the queries which we ask the decision maker, we explain how we incorporate these components into our multi-objective extension of top-two Thompson sampling, MOTTTS. We propose different variants, with different timings for querying the decision maker, to analyze the impact of the belief distribution over u.

If learning the utility function and learning the optimal arm are separate, disjoint processes, we can do one process followed by the other one. Thus, our first variant asks all the queries first, based on the initial belief distributions over arms, then learns the optimal policy using the learned utility function. We call this variant MOTTTS-start.

Analogously, our second variant first learns the belief distributions over the arms, then asks all the queries based on these learned beliefs. We call this variant MOTTTS-end.

Finally, to assess the impact of combining both processes together, we propose a third variant, where the timing of each query is spread out equally over the arm-pulling budget. We call this variant MOTTTS-interleaved.

4.4.6 Experiments

To assess the impact of intertwining the arm-selection process and the query process, we analyze the difference in performance of each algorithm variant on randomly generated bandits.

For our experiments, we assume the reward distributions are normally distributed, as this is often the case in the bandit literature (Chapelle & Li, 2011; Rhuggenaath et al., 2019;

Algorithm 9 MOTTTS with interleaved queries

Require: m, ν, α, β prior parameters on reward distribution, prior parameters on utility weight distribution, T^A pulling budget, T^q query budget, p success probability

```
1: H^A ← 0, empty history for each arm
2: H^q ← 0, empty interaction history
3: q ← T^A / T^q                                   ▷ query frequency
4: for t ∈ [0, T^A − 1] do
5:     if 0 = t mod q then                         ▷ ask a query every q steps
6:         H^q ← H^q ∪ Algorithm 8 (H^A, H^q)
7:     end if
8:     û ← x̄                                       ▷ mean of belief distribution over utility
9:     i ← Algorithm 5 (H^A, ν, α, β, û)           ▷ arm of the highest estimated mean w.r.t. û
10:    sample b ∼ B(p)
11:    if b = 1 then
12:        H_i^A ← H_i^A ∪ r_t,   r_t ∼ N(μ_i, σ_i) ▷ pull arm and add to history of arm-pulls
13:    else                                        ▷ sample second-best arm
14:        repeat
15:            j ← Algorithm 5 (H^A, ν, α, β, û)
16:        until i ≠ j
17:        H_j^A ← H_j^A ∪ r_t,   r_t ∼ N(μ_j, σ_j) ▷ pull arm and add to history of arm-pulls
18:    end if
19: end for
```

Roijers et al., 2017). We assume no correlation between the random variables of $\boldsymbol{\mathcal{P}}_\theta$, as this does not affect the multivariate mean. Thus, $\boldsymbol{\mathcal{R}}$ is defined as:

$$\boldsymbol{\mathcal{R}} = \{\{\mathcal{N}^0(\mu_0^0, \sigma_0^0), \ldots, \mathcal{N}^n(\mu_0^n, \sigma_0^n)\}, \ldots, \{\mathcal{N}^0(\mu_A^0, \sigma_A^0), \ldots, \mathcal{N}^n(\mu_A^n, \sigma_A^n)\}. \quad (4.12)$$

We define $\boldsymbol{\mu}_a = [\mu_a^0, \ldots, \mu_a^n]$, $\boldsymbol{\sigma}_a = [\sigma_a^0, \ldots, \sigma_a^n]$ as the multivariate mean, standard deviation for arm a, respectively (thus $\theta_a = \langle \boldsymbol{\mu}_a, \boldsymbol{\sigma}_a \rangle$).

Posterior distribution for normal distributions

MOTTTS keeps a belief distribution for each of the arms. As the reward distribution for each arm the considered MOMABs is a set of independent normal distributions, we keep a separate belief distribution for each normal. We define the posterior distribution based on (Murphy, 2007), to which we refer for its derivation.

We assume $\mathcal{P}_{\theta_a}$ is normally distributed, and consider the most generic case where both μ_a, σ_a are unknown. We would like to compute posterior distribution, i.e., the probability distribution $\mathcal{P}(\mu_a, \sigma_a \mid \mathcal{H}_{a,t})$ over the possible means, standard deviations, given the history $\mathcal{H}_{a,t}$. For this, we need the likelihood function $\mathcal{P}(\mu_a, \sigma_a \mid \phi_a)$. When both μ_a, σ_a are unknown, $\mathcal{P}(\mu_a, \sigma_a \mid \phi_a)$ is a normal-inverse-gamma distribution, written as $\mathcal{N}\Gamma^{-1}$, with parameters $\phi_a = (m_a, \nu_a, \alpha_a, \beta_a)$, where m_a represents the prior mean, ν_a the prior

sample size, α_a the prior shape and β_a the prior scale (Van den Burg, 2020):

$$\mathcal{P}(\mu_a, \sigma_a \mid \phi_a) = \mathcal{N}\Gamma^{-1}(m_a, \nu_a, \alpha_a, \beta_a). \tag{4.13}$$

Plugging this into Equation 4.4 results in

$$\mathcal{P}(\mu_a, \sigma_a \mid \mathcal{H}^A_{a,t}) \propto \mathcal{N}(\mu_a; \rho_{a,t}, \zeta_a)\Gamma^{-1}(\sigma_a^2; \frac{1}{2}N_t(a) + \alpha_a, \beta_{a,t}) \tag{4.14}$$

where

$$\begin{aligned}
\rho_{a,t} &= \frac{\nu_a m_a + N_t(a)|\bar{\mu}_{a,t}}{\nu_a + N_t(a)}, \\
\zeta_a &= \frac{\sigma_a}{\sqrt{N_t(a) + \nu_a}}, \\
\beta_{a,t} &= \beta_a + \frac{1}{2}s_{a,t} + \frac{N_t(a)\nu_a(\bar{\mu}_{a,t} - m_a)^2}{2(N_t(a) + \nu_a)},
\end{aligned}$$

with $\bar{\mu}_{a,t}$ the sample mean, $s_{a,t}$ the sample sum of squares:

$$\begin{aligned}
\bar{\mu}_{a,t} &= \frac{1}{N_t(a)} \sum_{r_i \in \mathcal{H}^A_{a,t}} r_i, \\
s_{a,t} &= \sum_{r_i \in \mathcal{H}^A_{a,t}} (r_i - \bar{\mu}_{a,t})^2.
\end{aligned}$$

Thus, sampling from our posterior distribution is performed in two steps. First, a variance σ_a^2 is sampled from the inverse-gamma distribution based prior parameters $\alpha_a, \beta_{a,t}$ and the number of pulls $N_t(a)$. Then, $\rho_{a,t}, \zeta_a$ are computed, using this sampled variance. Finally, we sample a mean μ_a from a normal distribution with mean $\rho_{a,t}$ and standard deviation ζ_a.

Experimental setup

So that learning the optimal policy is challenging, i.e., it is not possible to reliably find the optimal policy by randomly pulling arms, the generated bandits must satisfy some properties. First, so there can be different optimal arms depending on the utility function, we ensure a percentage of arms are non-dominated. In our experiments, we set this percentage to 40%.

Next, after sampling a random synthetic utility function by sampling from the $n-1$ simplex, we compare the difference in utility between the top-two arms. We ensure this difference is below a certain threshold, to reduce the number of ways to identify the best arm. In our experiments, we set this threshold to 0.1.

Finally, we ensure there is an overlap over the different arm-distributions by choosing high enough variance terms. In our experiments, the stochastic reward function for each arm

	MORobin-end	MOTTTS-start	MOTTTS-end	MOTTTS-interleaved	MOTTTS-cheat
BAI%	66.96%	72.70%	67.80%	75.07%	83.29%
WIN%	N/A	13.53%	0.11%	86.38%	N/A

Table 4.1: Comparison of MOTTTS with different timings for asking queries to the decision maker. For more insight on the performance of each timing strategy, we show a round-robin strategy as lower bound, and a "cheat" version of MOTTTS which uses the true utility function as upper bound. We show the best-arm-identification percentage (BAI%) for each strategy, based on 100000 experiments on 10000 MOMABs. We observe that MOTTTS-interleaved finds the optimal arm 75.07% of the time, compared to 72.70% and 67.80% for the other variants. We do not show the standard deviation, as the different MOMABs exhibit different properties. instead, we show the number of times each variant has the highest BAI%, and see that MOTTTS-interleaved has the highest BAI% on 86.38% of the MOMABs.

is represented by an independent normal distribution per objective, where the mean is sampled from a uniform distribution $\mu \sim \mathcal{N}(0, 1)$ and standard deviation is sampled from $\sigma \sim \mathcal{N}(0.05, 0.15)$.

To provide further insights on the generated bandits, we include additional baselines that serve as upper and lower bounds on the probability of identifying the optimal arm. As an upper bound, we assume knowledge of the utility function, and apply TTTS on the utility of the sampled multi-objective rewards, which is equivalent to MOTTTS without having to learn the utility function. We call this upper bound MOTTTS-cheat.

As a lower bound, we use a round-robin strategy, i.e., the arm-pulling budget is split equally for each arm (see Section 4.3.1). After pulling each arm the same number of times, all queries are asked to learn the utility function. This strategy avoids pulling arms in a smart way and does not take advantage of the learned belief distributions over arms.

We generate a total of 10000 MOMABs, and execute each algorithm 100000 times on each MOMAB. Each of the random MOMAB has 10 different arms, and 2 objectives. We set the total budget for arms to 40 and the total budget for queries to 5.

Results

Results are displayed in Table 4.1. For each algorithm, we report the best arm identification percentage (BAI%) averaged over all MOMABs. Since the generated MOMABs are different, it is more difficult to find the optimal arm for some than for others. This means the BAI% inherently varies across MOMABs. As such, we did not find any insight in including the standard deviation. Instead, we report the percentage of MOMABs for which each query-timing outperforms the others in terms of BAI% (i.e., WIN%[1] in the Table).

First, we notice that, as expected the round-robin strategy performs worst, as it does not use targeted exploration. Second, we observe that knowing the utility function does indeed result in better performances, as the MOTTTS-cheat upper-bound baseline performs

[1]These percentages do not exactly sum to 100, due to the rare occurrences where performance is equal.

best. Next, we observe that asking all queries before pulling arms results in a better performance than asking all queries after having used up the pulling budget. Since, at the start of training, we use uninformative belief distributions, our query-selection strategy (see Section 4.4.4) samples different utility functions, and random vectorial rewards, resulting in diverse queries. While these queries might not be realistic, they allow to narrow down the range of possible utility functions, resulting in a more reliable ranking of arms during the arm-selection steps. In contrast, having no information on the utility function means each non-dominated arm is potentially optimal. Thus, for all these arms, an accurate belief distribution is required, resulting in the pulling budget being split across more arms than necessary. Even if, afterwards, the queries are realistic, the uncertainty on the belief distributions over arms might be too high to accurately select the best arm. Indeed, we observe that the performance of MOTTTS-end is similar to the lower-bound, MORobin-end, as the exploration has been spread over too many arms.

Across all the query-selection timings, MOTTTS-interleaved performs best. This indicates that improving knowledge over the utility function and improving knowledge over the search space of the policy are intertwined processes. As an additional analysis, we observe using the WIN% that on 86.38% of the MOMABs, MOTTTS-interleaved has a higher BAI% than both MOTTTS-start and MOTTTS-end. This supports our conclusion about the intertwined processes.

Finally, although MOTTTS-interleaved has the highest BAI% across the query-selection variants, there is still a large gap with the upper-bound performance, which reaches 83.29%. We believe that, by further optimizing the timing at which the queries are asked, we can close this gap.

4.5 POMCP for query optimization

We have shown that learning the utility function is tied with learning the optimal policy, as MOTTTS-interleaved learns to identify the best arm significantly more often than its non-intertwined counterparts (MOTTTS-start and MOTTTS-end). Thus, we argue that we can optimize the timing of querying the decision maker, as there is still a gap in performance when executing MOTTTS-cheat (i.e., with a known utility function).

We propose, to the best of our knowledge, the first algorithm for best arm identification in the multi-objective setting. Our algorithm takes inspiration from Partially Observable Monte-Carlo Planning (POMCP) (Silver & Veness, 2010), which effectuates Monte-Carlo sampling to break the curse of dimensionality of large search-spaces. Moreover, POMCP can cope with partial observability of MDPs, by keeping and updating beliefs over states. This makes POMCP compatible with our setting, as we keep belief distributions over the arms and utility function, which are updated with each pull, and query, respectively. Finally, since POMCP is a planning algorithm based on tree-search, it is made for episodic settings with a finite number of timesteps. This is the case for our best-arm identification setting, where the pulling budget and query budget define the number of timesteps that can be executed.

POMCP is an online algorithm for action-recommendation. In its essence, it is an extension of Monte-Carlo tree search (MCTS) (Coulom, 2007b) for partially observable MDPs

(POMDPs). Although, contrary to MDPs and POMDPs, MOMABs do not have states, we will see in Section 4.5.2 that we can use the same concept behind these algorithms differently in MOMABs, by considering the whole process towards finding the best arm as a sequential decision process.

We first explain the main idea behind MCTS, as it is essentially the same as for POMCP. At each timestep t, MCTS performs a number of simulations, or *rollouts*, using a model of the environment. All rollouts start from the current state s_t. Based on these simulated rollouts, it estimates Q-values $Q(s_t, a)$ for each action a of s_t. It then recommends executing the action with the highest estimated Q-value in the environment. This leads to a new state s_{t+1}, at which point the process repeats: recommending an action to execute in s_{t+1}, based on simulated rollouts starting from s_{t+1}. The main particularity of POMCP, compared to MCTS, is that POMCP does not actually *know* the state s_t (or s_{t+1}) it is currently in. It has only a limited view on the state, and a belief distribution $\mathcal{P}$ on what this state could be. For each rollout, POMCP samples an estimated state $\hat{s}_t$, and starts the simulation from there. The recommended action is then based on the aggregated Q-value estimates over all $\hat{s}_t$ samples.

4.5.1 Simulating rollouts

Recommending an action is based on simulated rollouts. To identify with the highest confidence possible which action to recommend, the rollouts use a targeted exploration of the available model. This is done by building a tree of the possible action-sequences, and following the most promising branches of this tree.

Initially, our tree consists of a root-node, corresponding to the belief distribution $\mathcal{P}_t$ over the current state s_t (we will see that in our setting, we will consider a "state" as the histories of pulls and queries). This *belief-node* has one child-node per possible action, called *action-node*. These action-nodes have no child-nodes yet, however they will be created later on. Each node keeps track of the number of times it has been visited (the *visitation count*), as well as the average return of all rollouts passing through that node (the Value of that node).

A rollout is split in 4 phases. In the first phase, we walk down our current tree. At each belief-node, we select an action-node based on the UCB (see Section 4.3.2), as is done in the original MCTS and POMCP algorithms. We execute the action in the model of the environment, leading us to the belief distribution over the next state. If this belief does not correspond to a child-node of the selected action-node, we go to the next phase. Otherwise, we walk to that child-node, and repeat the process.

Thus, the second phase starts from an action-node that leads to a belief that has not been encountered before. We create a new belief-node for this belief, and add it as a child of the current action-node. Moreover, similar as for the root-node, we add one-child node per possible action to the newly created belief-node. The second phase is thus a node-creation phase. Since this occurs at every rollout, each rollout creates one belief-node. The size of the tree is thus proportional to the number of rollouts performed at each timestep.

Next, the third phase starts from the newly created node, and executes a fixed policy until reaching a final state in the model of the environment. We then assess the performance of

the rollout using a scoring function (i.e., the rollout's return).

Finally, the return is backpropagated through the tree, updating the visitation count and Q-value of each traversed node during this rollout.

4.5.2 Transition model of MOMABs for simulated rollouts

POMCP requires a model of the environment to make simulations. We propose to create a such a model, entirely based on our belief distributions over arms and utility function.

The length of an episode (and thus the maximal depth of the search tree built by POMCP) is defined by the sum of the query budget and pulling budget. At each timestep, we can either pull an arm, or query the decision maker. The first $a_1, \ldots, a_A$ actions pull the corresponding arm $a_1, \ldots, a_A$. The last action queries the decision maker, using the querying strategy explained in Section 4.4.4. The number of actions is thus $A+1$. However, if either the pulling budget or the query budget is 0, then we can only query, or pull, respectively.

Of course, these actions are executed in the model, as such we need to define what happens when we pull an arm or decide to query the decision maker. For each new rollout, we sample a virtual MOMAB, based on our belief distributions, by sampling an estimated mean, standard deviation $\hat{\boldsymbol{\mu}}_a, \hat{\boldsymbol{\sigma}}_a \sim \mathcal{P}(\cdot \mid \mathcal{H}^A_{a,t}), a \in \mathcal{A}$ for each arm, and sampling a utility function $\hat{u} \sim \mathcal{P}(\cdot \mid \mathcal{H}^q_t)$. When pulling an arm a, we sample a reward from the virtual MOMAB $\boldsymbol{r} \sim \mathcal{N}(\hat{\boldsymbol{\mu}}_a, \hat{\boldsymbol{\sigma}}_a)$. Our belief distribution over a is then updated using $\boldsymbol{r}$. Analogously, queries are evaluated on $\hat{u}$, and our belief distribution over the utility function is updated accordingly. At the end of the rollout, we can then evaluate our final belief against the virtual MOMAB, and thus provide a score measuring the rollout's quality. By using a different virtual MOMAB for each rollout, we ensure that the recommended action at the root-node is the best one, for all potential states covered by our belief distribution.

Using our proposed model comes with some challenges. Mainly, the quality of the estimated Q-values at the root-node are significantly impacted by the branching factor of the same tree, as a larger branching factor requires exponentially more simulations. We note 2 different branching factors, one for the belief-nodes, and another for the action-nodes. Each belief node has one child for each possible action. Thus, the branching factor increases with the number of possible arms. Hence, we expect a decrease of performance for MOMABs with a large number of arms. This is a known problem for tree-search approaches such as MCTS and POMCP, and multiple approaches have been proposed to cope with large action-spaces, typically by keeping a small subset of candidate-actions (Chaslot et al., 2008; Couëtoux et al., 2011; Coulom, 2007a; Gelly & Silver, 2011).

Analogously, action-nodes have a branching factor that depends on the number of subsequent beliefs encountered after executing the action tied with the action-node. However, pulling an arm in our proposed model results in a continuous reward-vector. Thus, each time we enter an action-node and execute its action, the sampled reward will be different, resulting in a different belief distribution and associated belief-node. As such, the branching factor for belief nodes is infinite, bounding the tree-depth to 3: the root node, the actions executable from the root node, and finally an endlessly growing number of next-belief-nodes. To ensure a meaningful tree-search, we need to cope with this infinite branching factor.

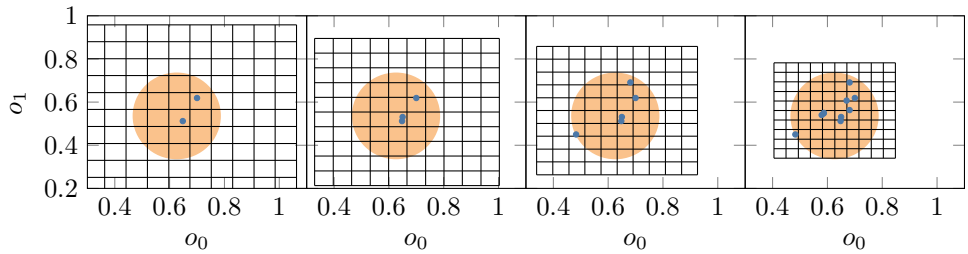


Figure 4.4: Example of produced bins, depending on the number of samples. The orange region shows an unknown reward distribution from which we sampled $2, 3, 5, 10$ times, from the leftmost plot to the rightmost plot, respectively. As the number of samples increases, our belief distribution becomes more precise, which means the bins better match the reward distribution.

4.5.3 Aggregating belief-nodes together

A straightforward way of dealing with continuous state-spaces is to discretize the states. For our setting, this amounts to splitting the range of possible reward values in a fixed number of bins. We can decide on the branching factor by choosing the number of bins. A higher number of bins increases the accuracy of the discretization, at the cost of a higher branching factor.

We use an adaptive binning mechanism, based on the belief distribution over the arm we are currently pulling. As the belief distribution becomes more accurate, due to the additional samples, then the bins become increasingly precise. Since, in our setting, the reward-distributions follow a Normal distribution, its conjugate prior is a Normal-gamma distribution (see Equation 4.14). We use the estimated mean and standard deviation from the conjugate prior to produce the boundaries between each bin.

Concretely, we compute a range of possible values $[\mu - c\sigma, \mu + c\sigma]$, where c is a constant defining up to how many standard deviations we can be away from the estimated mean, that we partition equally between the chosen number of bins. We set, using prior hyperparameters $\alpha = \frac{1}{2}, \beta = 0.1$:

$$\mu = \hat{\mu}, \text{with } \hat{\mu} \text{ the sample mean,}$$
$$k = \alpha + \frac{N}{2}, \text{with } N \text{ the number of samples,}$$
$$\tau = (\beta + \frac{N}{2}\hat{\sigma}^2)^{-1}, \text{with } \hat{\sigma} \text{ the sample standard deviation, assuming a zero-prior,}$$
$$\sigma = \sqrt{\frac{1}{k\tau}}, \text{since } k\tau \text{ is the mean of a Gamma distribution.}$$

Figure 4.4 shows an illustration of our binning method depending on the number of samples, using 10 bins per objective-dimension. It displays, in orange, the bivariate normal distribution (up to 2 standard deviations) from which the samples (in blue) are drawn. The

grid (in black) represent the different bins. As the number of samples increase, so does the confidence of the belief distribution in its estimation of the mean and standard deviation. The bins increasingly concentrate around the true distribution.

Our adapted binning allows for an automatic segmentation of the sampled rewards. However, the number of bins required to split the state-space increases exponentially with the number of objectives. Still, we argue that, when the number of objectives are limited, binning is a reasonable approach, as it is conceptually simple and adaptive to each individual belief distribution.

4.5.4 Evaluating rollouts

Since we can cope with the infinite branching of action-nodes, we can execute simulated rollouts that each will increase the size and depth of the search tree. At the end of a rollout, we need to assess its quality. Since our aim is to provide the best arm within a fixed budget, we would like our algorithm to recommend the action that maximally increases our chances to find this best arm. As such, our scoring function should reflect this.

Since we can best assess the quality of a rollout after it has been executed, and the score is backpropagated through the tree, we only provide a score at the end of each simulation, after having used all the budget. This score is computed using the posterior belief over arms and over the utility function, since they define our estimate over the best arm.

To assess the importance of the scoring function compared to the number of rollouts, we have analyzed 2 alternative scoring mechanisms. As a first scoring function, we compute our estimated utility of each arm, by using the estimated mean of the posterior belief over the utility function and the estimated mean of each arm. If the arm with the highest estimated utility, i.e., the proposed best arm, matches the best arm from the simulated bandit generated at the root node of our tree, the simulation is considered a success, and returns a score of 1. Otherwise, the simulation is considered a failure, and the returned score is 0:

$$\hat{a}^* = \arg\max_{a \in A} \hat{\boldsymbol{\mu}}_u^\top \hat{\boldsymbol{\mu}}_a, \tag{4.15}$$

$$\mathcal{R}_1 = \begin{cases} 1 & \text{if } \hat{a}^* = \tilde{a}^* \\ 0 & \text{otherwise.} \end{cases} \tag{4.16}$$

This is the same way we would recommend the best arm to the real decision maker (and not the simulated utility function in our algorithm). Since each action-node keeps track of the average score from all simulations passing through that node, the scores of the action-nodes at the root represent the probability of recommending the best arm to the decision maker.

Not only is this scoring function aligned with the goal of our problem setting, it has also the advantage of being computationally inexpensive, which allows us to spend the computational budget on additional simulations, thus improving the accuracy of the estimated probabilities at the root's action-nodes. However, due to its binary nature, this

scoring function is sparse. Moreover, since the success depends on our initial belief over the MOMAB, it can be noisy. Thus, many simulations are required for accurate best arm identification probabilities at the root node.

The first scoring function requires many simulations for accurate best arm identification probabilities at the root node. When the branching factor becomes too large, this can become an issue. Thus, we envisage a second, less sparse and more informative scoring function. We take inspiration from TTTS which, to identify the best arm, aims to discriminate the best and second-best arm as much as possible. In a similar fashion, we estimate the confidence of our posterior belief at the end of the simulation in recommending the best arm.

To compute this confidence score, we sample weights and arm-rewards multiple times from our posterior belief over the MOMAB. For each arm, we count the number of times it is recommended as best. This gives us a recommendation percentage for each arm. The returned score is the highest recommendation percentage.

Since the recommendation percentage depends on the number of times we sample from our posterior belief, it is more computationally expensive. However, it might be a beneficial trade-off depending on the properties of the bandit (e.g., number of arms, number of objectives).

4.5.5 Experiments

We evaluate our method on random MOMABs, generated in the same fashion as for our MOTTTS experiments (see Section 4.4.6). We generate 30 MOMABs, and perform 1000 experiments on each MOMAB. Furthermore, we compare our method against MOTTTS-interleaved, the top performing baseline, which ask queries at regular, fixed intervals. We use 100 particles equally-spaced over the $n-1$ simplex, initialized with uniform probability-weights. We set the UCB exploration factor β to 0.1.

Results are shown in Figure 4.5. We show MOPOMCP using the 2 scoring functions. MOPOMCP$_{s=1}$ uses the binary scoring function, while MOPOMCP$_{s=1000}$ estimates the posterior confidence with 1000 samples. We analyze the effect of the number of rollouts on MOPOMCP's performance, by repeating the experiments with an increasing number of rollouts. We note that, since MOPOMCP$_{s=1}$ uses a less computationally expensive scoring function than MOPOMCP$_{s=1000}$, it can perform more rollouts for similar wall-times. In fact, the number of rollouts for similar wall-times is one order of magnitude higher for MOPOMCP$_{s=1}$ than for MOPOMCP$_{s=1000}$. This is why we perform experiments with $10^4, 10^5, 10^6$ rollouts for MOPOMCP$_{s=1}$, and $10^3, 10^4, 10^5$ rollouts for MOPOMCP$_{s=1000}$.

In general, the BAI% increases with the number of rollouts. Provided enough rollouts, MOPOMCP outperforms the MOTTTS-interleaved baseline, regardless of the scoring function. However, for similar wall-times, MOPOMCP$_{s=1000}$ systematically outperforms MOPOMCP$_{s=1}$. Thus, it seems that using a more informative scoring function has a higher impact on performance than increasing the number of rollouts. MOPOMCP$_{s=1000}$ with 10^5 rollouts reaches a BAI% of 78.38%, compared to 74.48% for MOTTTS-interleaved and 85.26% for the upper bound, MOTTTS-cheat. Since we can maximally improve the BAI% by 10.78% compared to the baseline, the 3.9% improvement by MOPOMCP$_{s=1000}$

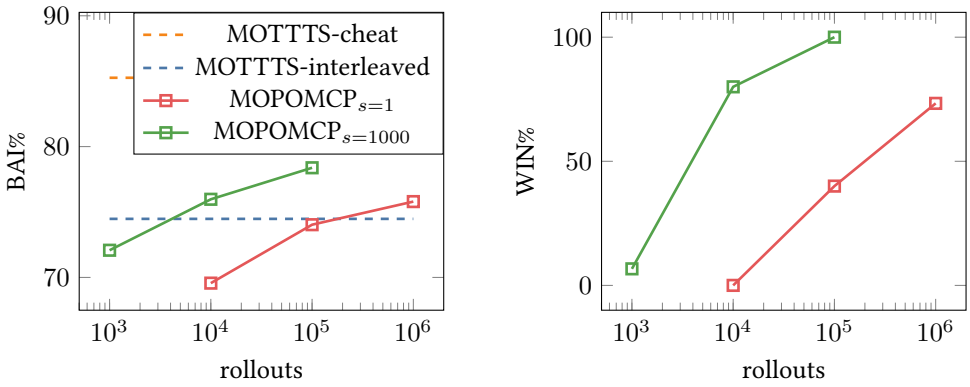


Figure 4.5: Best-arm-identification percentage (BAI%) of MOPOMCP depending on the number of rollouts. MOPOMCP$_{s=1}$ uses the binary scoring function, while MOPOMCP$_{s=1000}$ estimates the posterior confidence with 1000 samples. We note that MOPOMCP$_{s=1}$ needs one order of magnitude more rollouts than MOPOMCP$_{s=1000}$ to reach similar performance in BAI%. However, since the binary scoring function is less expensive to compute, the wall-time is similar. We observe that, given enough rollouts, the timing of the queries proposed by MOPOMCP results in a higher BAI% than the MOTTTS-interleaved baseline. Moreover, for similar wall-time, using the second scoring function results in a higher BAI% than using the binary scoring function.

represent a substantial increase in performance. Moreover, looking at the corresponding WIN% value on the right plot of Figure 4.5, we see that MOPOMCP$_{s=1000}$ has a higher BAI% than MOTTTS-interleaved on 100% of the generated MOMABs, showing that it reliably optimizes the timing of queries, regardless of the MOMAB's properties. In contrast, MOPOMCP$_{s=1}$ with 10^6 rollouts has a higher average BAI%, but is better than MOTTTS-interleaved on 73.33% of the generated MOMABs. Moreover, we expect the WIN% and BAI% to increase as we increase the number of rollouts. Since the best-arm-identification setting is not necessarily an online setting, we expect this to be possible depending on the problem at hand. Thus, our proposed MOPOMCP algorithm learns different timings depending on the MOMAB, allowing it to more efficiently pull arms with respect to the estimated utility, resulting in a higher chance of finding the best arm than when using a fixed timing for queries.

4.6 Related work

While, to the best of our knowledge, we are the first to propose an algorithm for best arm identification in the multi-objective setting, this setting is related to different areas of work.

MOMABs (Drugan & Nowe, 2013) have been studied to learn the set of Pareto optimal policies (Garrido-Merchán & Hernández-Lobato, 2019; Laumanns & Ocenasek, 2002). Yahyaa and Manderick (2015) and Yahyaa et al. (2014) assume Bernoulli distributions over the rewards, and additionally aim for a fair pulling of all Pareto-efficient arms. These methods minimize the regret using a multi-objective performance metric. For example, Daulton et

al. (2020) learn the set of Pareto-efficent arms using a fast approximation of the expected hypervolume improvement, while Belakaria et al. (2020) select the action that maximizes information gained about the Pareto front. Additionally, R. Zhang and Golovin (2020) learn the Pareto front by sampling different scalarization functions, based on the hypervolume.

In contrast to these methods that do not assume knowledge over the utility function, MOMABs have been used for known utility functions in the context of constrained optimization, either with linear constraints (Kagrecha et al., 2023), or non-linear constraints (Sui et al., 2015, 2018). However, they do not necessarily take an explicitly multi-objective approach, instead optimizing a single objective, subject to predefined constraints (Sui et al., 2015, 2018). Constrained optimization for MABs has been used in the context of clinical trials, by identifying dosages that satisfy toxicity constraints (C. Shen et al., 2020).

Instead of constraints, another approach is to provide an order of preferences over the different objectives, and incorporate that into a multi-objective Bayesian optimisation framework (Abdolshah et al., 2019).

More closely related to our setting, Paria et al. (2020) minimize the regret of MOMABs by randomly sampling utility functions. The sampling strategy can incorporate prior knowledge over the preferences of the decision maker, thus fine-tuning the set of optimal arms. While they do not incorporate a way to choose or update this prior, they argue their method is compatible with potential updates of information over preferences during the optimization process.

One other work that considers interactive preference learning for MOMABs is (Astudillo & Frazier, 2020). Like in our work, they incorporate pairwise relative queries, and they use parametric utility functions. Moreover, on top of linear utility functions, they consider quadratic and exponential utility functions. However, they focus on regret minimization, and ask queries at fixed intervals, like ITS (Roijers et al., 2017).

Finally, preference learning has been considered in RL (Wirth et al., 2016, 2017). However, while these methods use relative queries, they are based on partial trajectories, or states, instead of multi-objective trade-offs (Christiano et al., 2017; Ibarz et al., 2018). Interestingly, (Ziegler et al., 2019) consider both pairwise relative preferences and ranking preferences, by ordering 4 samples according to their preference.

4.7 Discussion

In this chapter, we have focused on the interactive decision support scenario (see Section 1.2). We have formalized the interactive setting as a MOMAB, in which we can query the decision maker. We propose MOTTTS with interleaved queries as a strong baseline for the best arm identification problem, demonstrating that learning the policy and learning the utility function are intertwined processes. Furthermore, we propose to use particle filtering as a belief distribution over linear utility functions, and show that this representation is more reliable than using Bayesian logistic regression.

Our experiments show that there is, on average, a large gap in performance between algorithms that learn the utility function, and the multi-objective variant of TTTS that has access to the utility function. By optimizing the timing at which we ask queries, we argue

we can close this performance gap. We propose, MOPOMCP, a Bayesian approach towards query-timing optimization for best-arm identification. Given enough simulated rollouts, MOPOMCP can accurately estimate which action maximally improves its belief over the best arm in terms of utility. Moreover, MOPOMCP copes with relative queries, which are more suited towards human decision makers.

We see many possible avenues for future work. First, we focused on linear utility functions in this chapter, which is a more restrictive setting than using non-linear utility functions. We can extend our work towards non-linear utility functions by using non-linear parametric scalarization functions, such as Chebyshev scalarization (Equation 2.9), and use particle filtering for the function parameters, similarly as we have done for the linear case. However, like for the linear case, the number of particles required to cover the parametric space increases exponentially with the number of objectives. Alternatively, we can use Gaussian Processes (GPs) to model non-linear utility functions, as we have done in (Roijers, Zintgraf, et al., 2021). While the GPs we use in (Roijers, Zintgraf, et al., 2021) cope with relative preferences, they do not enforce monotonicity constraints. While monotonicity constraints cannot be enforced on GPs, some approximations exist (Chu & Ghahramani, 2005; Riihimäki & Vehtari, 2010; Zintgraf et al., 2018). Further research is thus necessary to integrate these with MOPOMCP.

Second, we aim to scale our experiments to the many-objective setting, by proposing an adaptive particle filtering method that resamples filters in more interesting regions of the weight space (Hol et al., 2006; T. Li et al., 2015). We can also scale our experiments to MOMABs with more arms, by using, e.g., progressive widening (Chaslot et al., 2008; Coulom, 2007a) on the actions in MOPOMCP's search tree.

Third, we can, on top of optimizing the timing of the query, optimize the queries themselves, by proposing queries that most discriminate part of the belief distribution over the utility function. We can take inspiration from active learning, which deals with interactive settings (Brochu et al., 2010; Eric et al., 2007; Zintgraf et al., 2018).

Finally, using belief distributions over the utility function can be incorporated into MORL, such that we learn a policy that either provides us with information about the utility, when being presented to the decision maker, or is optimal with respect to our belief distribution over the utility function.

Chapter 5

Learning Pareto efficient policies for non-linear utility functions

5.1 Introduction . . . 85
5.2 Reward Conditioned Policies . . . 87
5.3 Pareto Conditioned Networks . . . 87
5.3.1 Building the dataset . . . 88
5.3.2 Training the Network . . . 88
5.3.3 Policy Exploration . . . 89
5.3.4 Updating the Dataset . . . 90
5.4 Experiments . . . 91
5.4.1 Deep-Sea-Treasure . . . 93
5.4.2 Minecart . . . 94
5.4.3 Crossroad . . . 94
5.5 Scaling Up the Objective-Space . . . 96
5.6 Related work . . . 98
5.7 Discussion . . . 100

5.1 Introduction

Previous chapters focus on algorithms with some form of knowledge over the utility function. However, in many real-world problems, the utility function is unknown beforehand. This can happen, for example, when there are multiple stakeholders with different preferences and objectives, and it is not understood how to capture all of these in a single utility function. It might also be infeasible to interact with the decision maker and refine our

knowledge over the utility function. For example, in the case multiple stakeholders are involved, each interaction might result in multiple debates and negotiations before agreeing on a preferred solution. In this case, learning all non-dominated trade-offs and present them to the stakeholders, so these negotiations occur only once, a posteriori, might be preferable.

We refer to the third main category defined in Section 1.2: no-knowledge algorithms, where the utility function is unknown. As in the previous chapters, we only make minimal assumptions concerning the shape of the utility function. Finally, we aim to avoid unexpected behavior due to the stochasticity of the policy. For example, for the management of a hydroelectric power plant, the decision maker does not want to be presented with a policy that has a probability of completely draining the water reservoir even if that policy is optimal, as it would have catastrophic consequences for nearby towns (Hayes, Rădulescu, et al., 2021). Thus, we aim to learn deterministic policies. As explained in Section 2.6.3, even for non-linear utility functions, settings allowing stochastic policies result in the convex coverage set being the optimal coverage set. However, in the case of deterministic policies, one should learn the Pareto front, where the non-dominance criterion defines the membership of a solution to the Pareto front.

Combining unknown non-linear utility functions with deterministic policies results in a complex setting. The lack of the additivity property (see Section 2.4) means we cannot straightforwardly apply the Bellman equation. Moreover, we have seen in Chapter 3 that, even if the utility function is known a priori, single-objective algorithms are outperformed by dedicated multi-objective algorithms in terms of sample-efficiency and can be outperformed in terms of utility. Thus, outer-loop methods, which focus on optimizing one utility function at a time, cannot easily use single-objective sub-routines and need to incorporate these dedicated multi-objective algorithms as well. Additionally, many outer-loops methods execute these sub-routines independently. Even though optimizing for separate trade-offs often results in exploring separate parts of the state-space, similar trade-offs are usually characterized by similar behavior. Thus, at least for part of the policy execution, they encounter similar states. We believe the sub-routines could benefit from sharing experience with each-other and avoid spending resources on exploring the same regions of the state-space.

In this chapter, we propose a novel method, *Pareto Conditioned Networks* (PCN) (Reymond, Eugenio, & Nowè, 2022), that is able to efficiently learn the policies that belong to the Pareto front. PCN conditions a single neural network on the desired compromise, so that it outputs the policy predicted to achieve it. Our method is sample-efficient, as it feeds the experience used for different compromises into the same network, thus allowing to share experience across policies. This also means it does not need to learn a policy independently for each trade-off found in the Pareto front, which is an approach taken by several works in multi-objective reinforcement learning (MORL) (Parisi et al., 2014; Roijers et al., 2015). Moreover, we make minimal assumptions concerning the utility function as opposed to the often-used assumption of linear scalarization of the objectives in the MORL literature (Abels et al., 2019; R. Yang et al., 2019). Finally, our method is scalable with respect to the number of objectives, as opposed to many of the current state-of-the-art MORL methods, who often limit themselves to 2- or 3-objective problems (Hayes, Rădulescu, et al., 2021).

5.2 Reward Conditioned Policies

Our work is inspired by the Reward Conditioned Policies algorithm proposed by (Kumar et al., 2019; Schmidhuber, 2019). Using neural networks as function approximators in RL comes with many challenges. One of them is that the target (e.g., the optimal action of the policy) is not known in advance — as opposed to classical supervised learning where the ground-truth target is provided. As the behavior of the agent improves over time, the action used as target can change, often leading to hard-to-tune and brittle learners (Fu et al., 2019; Mnih et al., 2015).

Instead of trying to continuously improve the policy by learning actions that should lead to the highest cumulative reward, Reward Conditioned Policies flips the problem, by learning actions that should lead to any *desired* cumulative reward (be it high or low). In this way, all past trajectories can be reused for supervision, since their returns are known, as well as the actions needed to reach said returns. We can thus train a policy that, conditioned on a desired return, provides the optimal action to reach said return. By leveraging the generalization properties of neural networks, we can accumulate incrementally better experience by conditioning on increasingly higher reward-goals.

In our work, we condition our policy on a multi-objective return, such that we can execute policies to reach diverse points on the Pareto front. We propose a training regimen focused on increasing the current solution set uniformly across the whole objective-space to avoiding catastrophic forgetting.

5.3 Pareto Conditioned Networks

In this Section we introduce our main contribution, the Pareto Conditioned Networks (PCN) algorithm. The key idea behind our approach is to use supervised learning techniques to improve the policy instead of resorting to temporal-difference learning. As explained in Section 5.2, this eliminates the moving-target problem, resulting in stable learning.

PCN uses a single neural network that takes a tuple $\langle s, \hat{h}, \hat{\boldsymbol{R}} \rangle$ as input. They represent, for state s, the return $\hat{\boldsymbol{R}}$ that PCN should reach at the end of the episode, i.e. the *desired return* of the decision maker. The *desired horizon* $\hat{h}$, that says how many timesteps should be executed before reaching $\hat{\boldsymbol{R}}$. At execution time, both $\hat{h}$ and $\hat{\boldsymbol{R}}$ are chosen by the decision maker at the start of the episode.

PCN's neural network has a separate output for each action $a_i \in \mathcal{A}$. Each output represents the confidence the network has that, by taking the corresponding action, the desired return will be reached in the desired number of timesteps. We can draw an analogy with a classification problem where the network should learn to classify $(s, \hat{h}, \hat{\boldsymbol{R}})$ to its corresponding label a_i.

As with classification, PCN requires a labeled dataset with training examples to learn a mapping from input to label. Contrary to classification, however, the data present in the dataset is not fixed. PCN collects data from the trajectories experienced while exploring the environment (see Section 5.3.1). Thus, the dataset improves over time, as we collect better and better trajectories. In particular, since the ability of PCN to reach Pareto-dominating

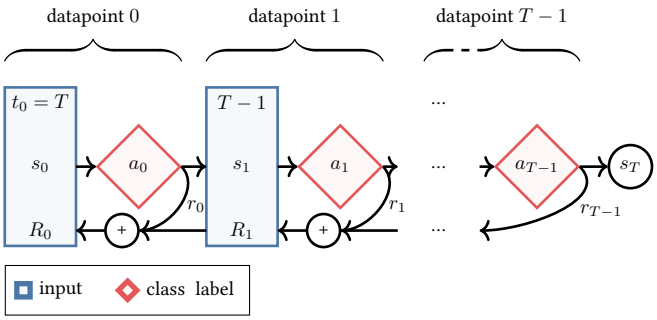


Figure 5.1: Conversion from a trajectory to labeled datapoints. For each timestep, we extract a single datapoint. The input (blue) is composed of the state at that timestep, and total return and number of timesteps until the end of the episode. The label (red) is the action taken at that timestep.

solutions depends on the data on which its network is trained, we want to keep only relevant experiences. We do so by limiting the size of the dataset and pruning it in such a way so to keep only tuples with $\hat{\boldsymbol{R}}$'s from different parts of the objective-space (see Section 5.3.4).

We collect new data for several episodes, after which we re-train the network with a number of batch updates from the new dataset (see Section 5.3.2). This improves the policies induced by the network, which in turn allows to gather better data for the next training batch.

5.3.1 Building the dataset

As mentioned before, each datapoint in PCN's dataset is comprised of an input $\langle s, \hat{h}, \hat{\boldsymbol{R}} \rangle$ and an output a. These are computed from observed transitions in the environment. As the dataset is empty at first, we execute a random policy on the environment for the first few episodes in order to collect a variety of trajectories.

After each episode is completed we store its trajectory. Then, for each timestep t of the trajectory, we know how many timesteps are left until the end is reached, i.e., the episode's horizon $h_t = T - t$. We can also compute the cumulative reward obtained from timestep t onward, i.e., $\boldsymbol{R}_t = \Sigma_{i=t}^{T} \gamma^i \boldsymbol{r}_i$. Since for this trajectory executing action a_t in state s_t resulted in return $\boldsymbol{R}_t$ in h_t timesteps, we add a datapoint with input $\langle s, \hat{h}, \hat{\boldsymbol{R}} \rangle = \langle s_t, h_t, \boldsymbol{R}_t \rangle$ and output $a = a_t$ to the dataset. In other words, when the observed return corresponds to the desired return in that state, then a_t is the optimal action to take. Figure 5.1 shows how a full trajectory is decomposed into individual datapoints.

5.3.2 Training the Network

The network architecture of PCN uses a separate embedding for the state and another one for the desired return and horizon. The desired return and horizon are concatenated together and multiplied by a scaling factor to normalize their values. They then pass

through a single fully connected layer followed by a sigmoid function. Similarly, the state-embedding also ends with a fully connected layer and sigmoid activation. Both layers have the same number of output nodes (we use 64 for all our experiments), which are combined together using the Hadamard product. The Hadamard product has been shown to make a more effective use of conditioning variables (E. Perez et al., 2018), and the sigmoid used on both outputs ensure that both embeddings are equally important. Finally, the resulting output passes through a multilayer perceptron that has a separate output node for each action, and a single hidden layer of 64 nodes with a ReLU activation function.

PCN trains the network as a classification problem, where each class represents a different action. Transitions $x = \langle s_t, h_t, \boldsymbol{R}_t \rangle, y = a_t$ are sampled from the dataset, and the ground-truth output y is compared with the predicted output $\hat{y} = \pi(s_t, h_t, \boldsymbol{R}_t)$. The predictor (i.e., the policy) is then updated using the cross-entropy loss function:

$$H = -\sum_{a \in \mathcal{A}} y_a \log \pi(a \mid s_t, h_t, \boldsymbol{R}_t), \tag{5.1}$$

where $y_a = 1$ if $a = a_t$ and $y_a = 0$ otherwise.

The network is re-trained periodically, but only after a set number of episodes (which is a hyperparameter that depends on the problem), to ensure that enough new experience has been collected and that the underlying dataset has been improved sufficiently.

5.3.3 Policy Exploration

As our dataset is composed of transitions collected from training experience, we can see that the quality of our dataset crucially depends on the quality of the executed trajectories. It is unrealistic to expect PCN to reliably produce trajectories with high-valued desired returns when it has only been trained on datapoints originating from random trajectories. Rather, we can expect PCN to produce trajectories with returns in the range of the ones from the current training data. Therefore, if we obtain trajectories reaching high returns, PCN will be able to confidently return high-return policies.

PCN leverages the fact that, due to the generalization capabilities of neural networks, the policies obtained from the network will still be reliable even if the desired return is marginally higher than what is present in the training data. In fact, they will perform similar actions to those in the training data, but lead to a higher return. Thus, we incrementally condition the network on better and better returns, in order to obtain trajectories that extend the boundaries of PCN's current coverage set.

More precisely, we randomly select a non-dominated return $\boldsymbol{R}_{nd}$ and its corresponding horizon $\hat{h}$ from the dataset. By randomly picking a non-dominated return from the entire coverage set we ensure equal chance of improvement to each part of the objective space. However, using $\boldsymbol{R}_{nd}$ exactly would induce the network to only replicate the already observed sampled trajectory so, as a second step, we choose a single objective o to improve upon. We then increase the desired value for that objective to obtain a new target return. PCN determines the magnitude of the increase by computing the standard deviation σ_o for the selected objective, using all non-dominated returns from the trajectories in the dataset. The magnitude is then sampled from the uniform distribution $U(0, \sigma_o)$ and added to $R_{nd,o}$

to form our desired return $\hat{\boldsymbol{R}}$. By restricting the improvement to at most σ_o, $\hat{\boldsymbol{R}}$ stays in the range of possible achievable returns and, by only modifying one objective at a time, the changes to the network's input compared to the training data are kept at a minimum.

With $\hat{\boldsymbol{R}}$ and $\hat{h}$ selected, PCN can condition its network and act during the training episode. At the start of the episode, $\langle s_0, \hat{h}, \hat{\boldsymbol{R}} \rangle$ results in executing a_0 and observing $\boldsymbol{r}_0, s_1$. PCN then updates the desired return and horizon such that they stay consistent throughout the episode: $\hat{\boldsymbol{R}} \longleftarrow \hat{\boldsymbol{R}} - \boldsymbol{r}_0$ and $\hat{h} \longleftarrow \max(\hat{h} - 1, 1)$. We ensure that $\hat{h}$ is at least 1 to avoid impossible desired horizons. PCN can then choose an action for s_1. This process is repeated until PCN encounters a terminal state. The trajectory is then added to the dataset, using the conversion to datapoints explained in Section 5.3.1.

To increase the range of observations in the environment during training, PCN samples actions from a categorical distribution with each action's confidence score corresponding to its probability of being sampled. This is done by using a softmax function on the network's output (Szandała, 2021). Note that at execution time — i.e., after the training process — we use a deterministic policy by systematically selecting the action with the highest confidence. This is because, as mentioned in Section 2.6.3, only deterministic policies are optimal when learning the complete set of Pareto-efficient policies.

5.3.4 Updating the Dataset

As it collects new experience, PCN needs to use it to train its network without forgetting about previous, relevant experience. Unfortunately, measuring relevance of data in our setting is non-trivial. We mainly care about the non-dominated solutions, since those are the ones that compose our coverage set, so ideally we would only keep their associated trajectories in our dataset. However, focusing solely on the current coverage set can lead to performance degradation if it is composed of too few V-values. This is because, during exploration (see Section 5.3.3), we will keep collecting very similar trajectories to the ones we already have. In turn, this will reinforce PCN's strategy to focus only on these few policies, disrupting the learning process. Thus, throughout training we keep a dataset of N trajectories, favoring trajectories that span different parts of the objective-space while removing highly clustered solutions. To do this we employ an additional metric, the *crowding distance*, that assigns a lower score to points with close neighbors. We then combine both the distance to the coverage set and the crowding distance in a single metric, which we use to prune less relevant points from the dataset.

We measure our preference for non-dominated V-values by computing for each solution its negative L2-norm distance, I_{l2}, to its closest non-dominated solution in the dataset. Non-dominated V-values thus score the highest with $I_{l2} = 0$.

$$I_{l2,i} = -\min \|p_i - p_j\|_2, p_j \in \hat{\Pi}, \tag{5.2}$$

where i is the index of the i-th solution in the dataset and p_j is a non-dominated point in the current coverage set $\hat{\Pi}$.

We measure the level of clustering of a V-value, I_{cd}, using the crowding distance (Deb et al., 2000). It assigns a score for each solution based on the distance between its neighbors in each dimension. Thus, V-values with close neighbors will have lower scores, while more

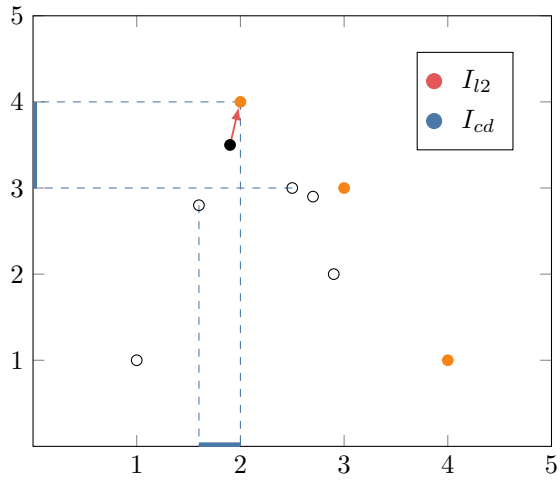


Figure 5.2: The I_{ds} metric for the solid black dot combines its negative L2-norm distance (red arrow) to its closest non-dominated neighbor (orange), and its crowding distance as the sum, for each dimension, of the max distance between a point's upper and lower neighbor (blue).

isolated points will have higher scores. Algorithm 10 shows how the crowding distance is computed in practice.

We then combine I_{l2} with I_{cd} in a single score metric, which we call *dominating score*, I_{ds}. We define it as:

$$I_{ds,i} = \begin{cases} I_{l2,i} & \texttt{if} \;\; I_{cd,i} > 0.2 \\ 2(I_{l2,i} - c) & \texttt{if} \;\; I_{cd,i} \leq 0.2 \end{cases} \tag{5.3}$$

Note that since I_{l2} corresponds to a negative distance, if a point is crowded — i.e., $I_{cd,i} \leq 0.2$ – we double the distance penalty. In addition, we add an additional small penalty c to crowded points to prune, to detect and prune duplicate points on the coverage set. Figure 5.2 shows an example on how to compute the L2-norm and crowding distances for a given solution set.

5.4 Experiments

Most state-of-the-art algorithms in MORL assume that the utility function can be expressed as a linear scalarization. This makes them unsuitable as relevant baselines, as their setting is different from ours. Thus, for all experiments, we compare PCN with 2 baselines that share our same assumptions on the utility function and that learn a set of policies that estimates the whole Pareto front. The first baseline, Multi-Objective Natural Evolution Strategies (MONES) (Parisi et al., 2017) uses a parametrized policy. It learns a distribution over the policy parameters such that sampling from this distribution produces a Pareto-efficient policy. Different samples produce different Pareto-efficient policies, each leading

Algorithm 10 Crowding Distance

Require: p points
Ensure: The crowding distance I_{cd}

```
for o ← 0 to n do
    s_i ← arg sort p_{.,o}
    for j ← 0 to len(p) do
        n_u, n_d ← s_{i_{j+1}}, s_{i_{j-1}}
        c_{s_{i_j},o} = p_{n_u} − p_{n_d}
    end for
end for
return I_cd ← Σ_o c_{.,o}
```

	Deep Sea Treasure	Minecart	Crossroad	Walkroom
learning rate	1×10^{-2}	1×10^{-3}	1×10^{-3}	1×10^{-2}
batch-size	256	256	1024	256
updates per iteration	10	50	50	10
exploration episodes	50	20	50	50
buffer size	200	50	50	$10n^3$

Table 5.1: The hyperparameters of Pareto Conditioned Networks, for each experiment.

to a different non-dominated return. The advantage of this method is that it can produce an infinite number of different policies. However, the main drawback is that we do not know the return of the sampled policy without executing it first.

The second baseline used is the Radial Algorithm (RA) (Parisi et al., 2014). RA trains a fixed number of independent policies. Each policy is trained using a policy gradient algorithm, where the gradients w.r.t. the different objectives are weighted together. Using different weights on these gradients produces distinct policies, each aiming for different regions of the objective-space. The main disadvantage of this approach is that every new policy is learnt independently, disregarding potentially useful experience encountered by other policies.

In contrast, our approach makes efficient usage of encountered experience as it learns a single network that, when conditioned on a desired return, produces a policy with predictable behavior.

When not mentioned otherwise, all results are averaged over 5 runs. The hyperparameters used are summarized in Table B.1. The code is publicly available online[1].

The results for all experiments are summarized in Table 5.2. Figures 5.3-5.6 show, for each environment, the coverage set found by each algorithm, with dominated solutions filtered out.

[1]https://github.com/mathieu-reymond/pareto-conditioned-networks

Algorithm 11 Pareto Conditioned Networks

Require: PCN network π_θ with buffer $\mathcal{B}_{\text{PCN}}$.

```
for n ∈ [N] do                                          ▷ fill buffers with random trajectories
    sample trajectory τ = ⟨s_{0:T}, a_{0:T}, r_{0:T}⟩ using random policy π_n
    for τ_i ∈ τ do
        ĥ, R̂ = T − i, Σ_{t=i}^{T} γ^{t−i} r_t                ▷ target horizon, return
        add τ_i, ĥ, R̂ to B_PCN
    end for
end for
while True do
    for m ∈ [M] do                                      ▷ policy updates
        update π_θ using B_PCN
    end for
    prune B_PCN using {R_τ | ∀τ ∈ B_PCN}
    select ĥ, R̂ based on B_PCN                          ▷ optimistic targets
    for n ∈ [N] do                                      ▷ exploration
        sample τ = ⟨s_{0:T}, a_{0:T}, r_{0:T}⟩ using π_θ with ĥ, R̂
        for τ_i ∈ τ do
            add τ_i, T − i, R_i to B_PCN
        end for
    end for
end while
```

We experimentally validate our method on three multi-objective benchmarks:

- Deep-Sea-Treasure (Vamplew et al., 2011), a well-known 2-objective grid-world problem (see Section 5.4.1),
- Minecart (Abels et al., 2019), a 3-objective problem with a continuous state-space (see Section 5.4.2),
- Crossroad, a novel 2-objective traffic environment, with high-dimensional pixel-like states (see Section 5.4.3).

In addition, in Section 5.5 we propose an additional novel high-objective environment, Walkroom, where we test our algorithms with up to 9 objectives.

5.4.1 Deep-Sea-Treasure

Deep-Sea-Treasure (DST) is a well known environment in multi-objective literature (see Section 3.8.1 for further details on the environment), where the agent controls a submarine in search for treasure hidden in the depth of the ocean. The agent must balance a trade-off between fuel consumption and treasure value. Navigating consumes fuel, but deeper treasures are worth more than shallow ones.

DST is a fairly small environment, which allows us to compute Pareto front analytically. It is composed of 10 different points, which form a concave front.

	hypervolume		
	PCN (ours)	MONES	RA
DST	$\mathbf{22845.40 \pm 19.20}^*$	17384.83 ± 6521.10	22437.40 ± 49.20
Minecart	$\mathbf{197.56 \pm 0.70}^*$	123.81 ± 23.03	123.92 ± 0.25
Crossroad	$\mathbf{539.53 \pm 6.27}^*$	429.09 ± 27.47	466.02 ± 31.23
	I_ε indicator		
	PCN (ours)	MONES	RA
DST	$\mathbf{0.039 \pm 0.087}^*$	0.687 ± 0.222	0.667 ± 0.000
Minecart	$\mathbf{0.271 \pm 0.087}^*$	1.596 ± 0.889	1.000 ± 0.000
Crossroad	$\mathbf{0.247 \pm 0.172}^*$	0.660 ± 0.200	0.408 ± 0.039

Table 5.2: Mean and standard deviation of hypervolume and ε-indicator across all 5 runs for all algorithms. For hypervolume, higher is better. For I_ε, lower is better. Best results are highlighted in bold and with an asterisk.

Figure 5.3 shows the discovered points of the Pareto front by PCN and our two baselines. Only PCN is able to fully recover the Pareto front. RA only discovers the extrema, and is unable to find any of the points in the concave part of the front. While MONES performs slightly better, the number and value of points it discovered was highly variable depending on the run, explaining the high variance seen in Table 5.2.

5.4.2 Minecart

Minecart is a complex environment with a continuous state-space (Abels et al., 2019). Starting at a base station, the agent controls a cart with the goal to extract ores from mines scattered in the environment, and sell them back at its base. The cart has a limited capacity, so it cannot be filled indefinitely. Finally, actions consume fuel, making the Minecart problem a 3-objective problem. For further details for this environment, we refer to Section 3.8.2. In our experiments, we use the 6-dimensional continuous state-space.

Because the cart has a limited capacity to store ores, the agent must decide the ratio of each ore present in the cart. This is why, as can be seen in Figure 5.4, the vast majority of policies discovered by PCN are laid out in a straight line: the sum of R_0 and R_1 equals 1.5 (which is the cart capacity). A few policies only partially fill the cart, saving a bit of fuel in the process. In comparison, the coverage sets discovered by both baselines only contain a small subset of the possible ore ratios, and they systematically consume more fuel than our method. Finally, Table 5.2 reports that PCN achieves a low variance in hypervolume across the different runs, which shows that PCN is consistent in finding these diverse and efficient policies.

5.4.3 Crossroad

We evaluate our proposed method on a novel traffic environment, Crossroad, developed using the SUMO framework (Lopez et al., 2018). In this environment, the agent controls the traffic lights at a busy intersection between two bidirectional roads, a horizontal one with two lanes and a vertical one with a single lane. The traffic flow on each lane is high,

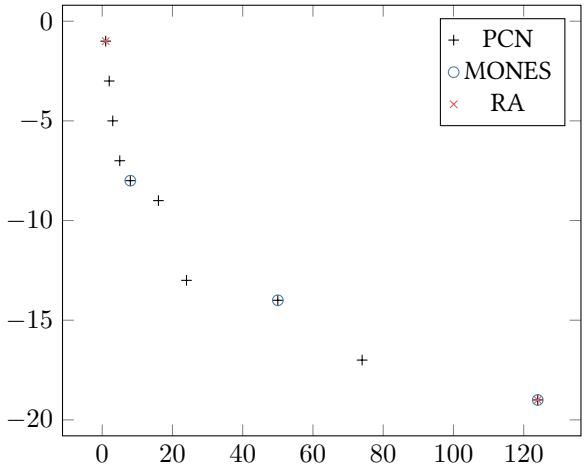


Figure 5.3: Best coverage sets for each algorithm in the Deep Sea Treasure environment. PCN is the only algorithm that recovered the full Pareto front.

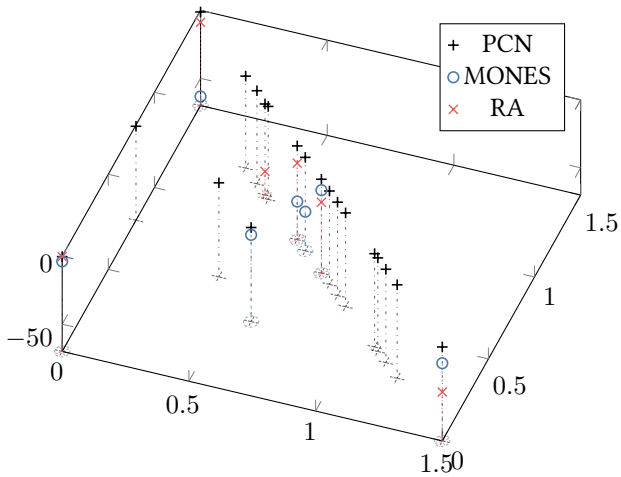


Figure 5.4: Best coverage sets for each algorithm in the Minecart environment. The straight shape of PCN's coverage set is due to the cart weight limit. PCN's coverage set fully dominates the ones from the baselines.

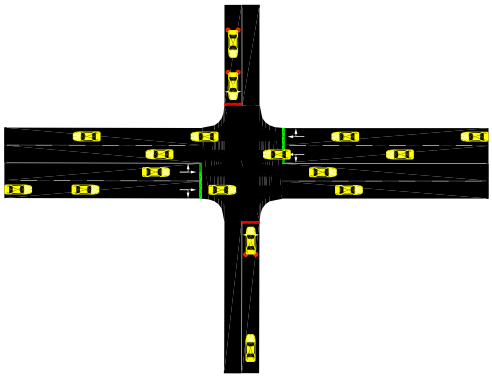


Figure 5.5: A visual representation of the Crossroad environment. The agent controls the traffic lights at the center of the intersection. Because one road has more lanes than the other, traffic flow is highest when its light is always green. This comes at the expense of a never-ending waiting time for the cars on the other road.

resulting in quickly growing waiting queues on the lane where the traffic light is red.

The agent can choose between 2 actions, either switching the lights or not. For a realistic traffic light system, the traffic lights first turn orange after the switching action has been made. Afterwards, they turn to the next phase (red or green). Actions during the orange phase are ignored, so that the lights cannot be immediately switched again.

The agent perceives a 2-dimensional grid of the top-down view of the intersection. Cells represent a section of a lane. A cell is filled with 0 if no car is present. If a cell contains a car, its value is the number of timesteps that the car has been on the lane. Thus, when a traffic light is red, all cells of the corresponding lane with a car increment their value by 1 at each timestep.

We consider 2 objectives: the first is the traffic flow, computed as the number of cars that leave the intersection. The second objective is the car waiting time, i.e., the number of timesteps a car has to wait before exiting the crossroad. Thus, favoring traffic on the horizontal road will favor the first objective, while alternating often will favor the second. The environment is depicted in Figure 5.5.

Figure 5.6 shows the coverage sets found by PCN and the two baselines. The points discovered by our method dominate the ones found by the baselines across nearly the whole objective-space. There is a single exception is the rightmost extremum, which corresponds to the policy that never switches the light, only allowing cars from the major road to cross the intersection. This policy is pretty simple, as it consists of always performing the same action, which explains why all methods find it.

5.5 Scaling Up the Objective-Space

Our experimental section shows that our proposed method significantly outperforms the baselines in several settings, with discrete, continuous and high-dimensional state-spaces.

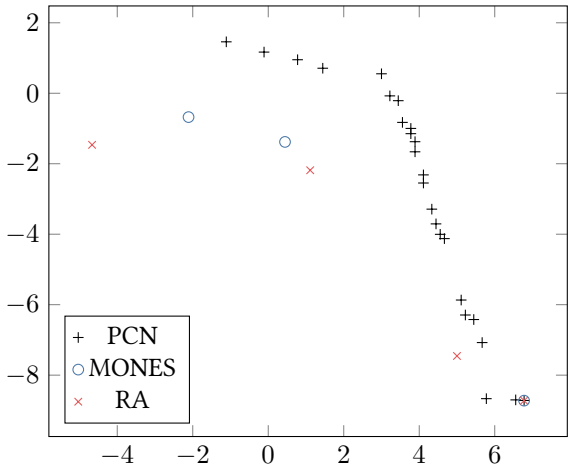


Figure 5.6: Best coverage sets for each algorithm in the Crossroad environment. PCN fully dominates the baselines but for the naive policy that never switches the traffic lights (bottom-right point).

It also empirically shows that the coverage sets discovered by PCN contain more non-dominated solutions than our selected baselines. Nevertheless, the vast majority of benchmarks used in MORL, including the ones used in our experiments, are limited to 2, sometimes 3 different objectives (Hayes, Rădulescu, et al., 2021).

To get a better understanding of the performance of PCN w.r.t. the number of objectives, we devise a synthetic environment, *Walkroom*, that can be instanced with an arbitrary number of objectives. Walkroom takes inspiration from Deep Sea Treasure, and is modeled as an n-dimensional grid-world in which the agent can move in every cardinal direction. Thus, the dimensions correspond to the number of objectives of the environment. The action space increases linearly with the number of objectives ($|\mathcal{A}| = 2n$). At each timestep, the agent receives a -1 reward for the objective corresponding to its moving dimension, and a 0 reward for all other objectives. There are no other rewards. Similarly to Deep Sea Treasure, Walkroom has a set of goal states positioned along an uneven border, so that reaching each goal state results in a different Pareto-efficient solution. The optimal policies are thus to go directly towards any of the border-positions, at which point the episode ends.

We evaluate each method in a set of randomly generated Walkroom environments, from 2 to 9 dimensions. We perform 20 runs for every algorithm, on every version of the environment ($n = 2, \ldots, 9$). Note that we do not plot the discovered coverage sets since this is not possible for $n > 3$. Instead, we show boxplots of the hypervolume and ε metrics computed on all runs. Since the Pareto fronts can be computed analytically when the environments are generated, the ε metrics give us an accurate representation of the coverage of each solution set found by each learning algorithm.

Results are summarized in Figure 5.7. RA performs poorly compared to the other algorithms across all metrics. This is because the number of policies that RA requires to cover

the objective-space increases exponentially with the number of objectives. Thus, training time must be split between more and more policies, which reduces their individual performance. For the same reason, we were unfortunately unable to compare in RA in environments with more than 5 objectives, due to its exponential computational costs.

For $n \leq 4$, MONES achieves similar or better scores than PCN in the I_ε metric. This is likely due to PCN missing some points from the Pareto front, which impact this metric significantly. However, we can see from the hypervolume and $I_{\varepsilon-mean}$ metrics that PCN generally performs better than MONES on the rest of the Pareto front. Surprisingly, for $n > 4$ MONES performs significantly worse than PCN. This might be because the parameter distributions of MONES are not able to explore efficiently in very large dimensional spaces.

5.6 Related work

Most of the recent work on MORL assumes an unknown, but linear utility function. However, this restricts the solution set that can be learned, as linear scalarization assumes the Pareto front to be convex. In the linear setting, the goal is to train an agent such that the optimal policy can be recovered for any preference weights. Roijers et al. (2015) propose Optimistic Linear Support (OLS), a generic method that iteratively selects different sets of weights and calls a single-objective learner as subroutine to find the corresponding optimal policy. Mossalam et al. (2016) extend this method for Deep RL. OLS has also been combined with Successor Features (Barreto et al., 2017) to form a convex coverage set (Alegre et al., 2022). This has then been further extended in (Alegre et al., 2023), using Generalized Policy Improvement (Barreto et al., 2020).

Another approach, taken by Barrett and Narayanan (2008) and Hiraoka et al. (2009), is to directly optimize on the coverage set without single-objective subroutine, by modifying the Bellman equation. Xu et al. (2020) combine evolutionary algorithms (Deb et al., 2000) with the actor-critic framework (Abdolmaleki et al., 2018; Fujimoto et al., 2018; Schulman et al., 2017) for continuous action-spaces. Contrary to PCN, these methods are unable to discover any V-values on the concave regions of the Pareto front (Das & Dennis, 1997).

Using conditioned networks has been explored in MORL, but again restricted to the linear scalarization setting. In (Castelletti et al., 2012), Fitted Q-Iteration (FQI) is extended to use a modified Q-network conditioned on *preference weights* instead of target returns. Similarly, Abels et al. (2019) use such a conditioned Q-network to extend Deep Q-Networks (DQN). Moreover, a similar network is used in (R. Yang et al., 2019), in combination with a multi-objective Bellman operator.

Our work also can be related to imitation learning, as it also uses supervised learning to learn a policy (Osa et al., 2018; Sun et al., 2018). However, imitation learning requires expert trajectories to train on, while PCN generates its own set of trajectories.

When the utility function can be any monotonically-increasing function, White (1982) adapt Value Iteration, while Moffaert and Nowé (2014) and Ruiz-Montiel et al. (2017) adapt tabular Q-learning to directly learn the Pareto front. A similar approach is taken by Wiering and De Jong (2007), which has been extended to model-based RL (Wiering et al., 2014). However, these approaches are limited to discrete low-dimensional state-spaces. Moreover, these methods assume a deterministic reward function, as the optimal policy cannot

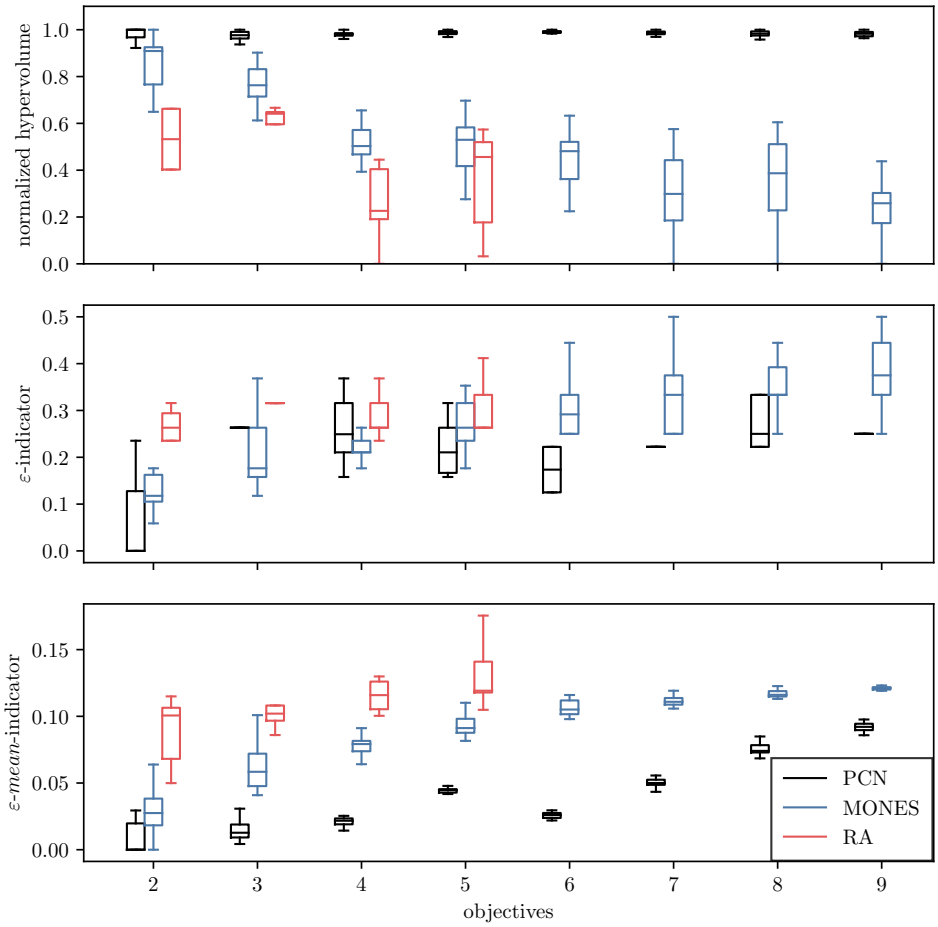


Figure 5.7: Box plots of normalized hypervolume, I_ε and $I_{\varepsilon-mean}$ in the Walkroom environments, from 2 to 9 objectives. We normalized the hypervolumes as their true values scale exponentially with the number of objectives (for $n = 9$, in the order of 10^9). For hypervolume, higher is better. For ε-indicator metrics, lower is better.

be reliably executed for stochastic rewards (Roijers, Röpke, et al., 2021). Alternatively, M. Agarwal et al. (2022) propose a framework to learn policies which maximizes a concave function, that is not necessarily monotone. We have extended tabular Q-learning to the deep MORL setting, with mitigated success on more complex environments (Reymond & Nowé, 2019).

Model-based approaches have also been investigated for MORL, by modifying MCTS to learn either the convex hull (Painter et al., 2020) or the Pareto front (D. Perez et al., 2014; W. Wang & Sebag, 2013). Since these approaches are model-based, they require to know the transition-function in advance. Alternatively, the model can be learned, an approach

taken for linear scalarization by Alegre et al. (2023) and Yamaguchi et al. (2019).

Related fields that have been adapted for MORL include meta-learning, where the meta-policy quickly specializes towards desired weights (Chen et al., 2019), constrained learning, where a subset of the objectives are seen as constraints (Huang et al., 2022), and distributional RL, where a distribution over actions is learned based on constraints per objective (Abdolmaleki et al., 2020).

Finally, Parisi et al. (2014) learn to reach the Pareto front using a modified policy gradient search, and Parisi et al. (2017) do this by modifying evolution strategies. Both algorithms make the same assumptions on the utility functions as PCN and are used as baselines in our experimental section (Section 6.4).

5.7 Discussion

We have presented a novel algorithm, Pareto Conditioned Networks, which is able to efficiently and effectively learn coverage sets in multi-objective sequential problems. PCN uses a single neural network to generalize experience across all possible multi-objective returns, learning coverage sets even in concave Pareto fronts.

We evaluated the empirical performance of PCN in several environments against state-of-the-art benchmarks. PCN was able to consistently obtain higher returns than the baselines throughout the whole objective-space, demonstrating its ability to exhaustively discover optimal coverage sets. In addition, PCN demonstrated its ability to learn the Pareto front even when dealing with a large number of objectives.

While PCN can work in continuous state-spaces, its network architecture is currently limited to discrete action-spaces. However, the main ideas behind PCN do not change for continuous action-spaces and PCN can be extended to continuous actions by changing the classification problem to a regression one. We use this idea in Chapter 6.

Chapter 6

Use-case: exploring the Pareto front of multi-objective COVID-19 mitigation policies using reinforcement learning

6.1 Introduction . . . 102
6.2 COVID-19 model and the MOBelCov MOMDP . . . 103
6.2.1 Stochastic compartment model for SARS-CoV-2 . . . 103
6.2.2 Interventions strategies . . . 106
6.2.3 A MOMDP for the compartment model . . . 106
6.3 Pareto Conditioned Networks for MOBelCov . . . 109
6.3.1 Training the network for continuous actions . . . 109
6.3.2 Coping with stochastic transitions . . . 110
6.4 Experiments . . . 110
6.4.1 Learned coverage set . . . 111
6.4.2 Impact of budgets on the coverage set . . . 112
6.4.3 R_{ARH} versus on R_{ARI} . . . 113
6.4.4 Robustness of policy executions . . . 116
6.5 Related work . . . 117
6.6 Discussion . . . 117

6.1 Introduction

Over the course of this thesis, we have developed MORL algorithms for various multi-objective settings, which we experimentally validated on a number of benchmark environments. In this Chapter, we apply and extend one of our methods to a real-world problem: mitigating epidemic outbreaks.

As shown by the COVID-19 pandemic, infectious disease outbreaks represent a major global challenge. To this end, understanding the complex dynamics that underlie these epidemics is essential. Epidemiological transmission models allow us to capture and understand such dynamics and facilitate the study of prevention strategies through simulation. However, developing efficient mitigation strategies remains a challenging process, given the non-linear and complex nature of epidemics.

Given that RL learns a policy through trial-and-error, without knowledge of the underlying dynamics of the environment it interacts with, it provides a methodology to automatically learn mitigation strategies in combination with complex epidemic models (P. J. K. Libin et al., 2021).

Previous research focused on optimizing policies with respect to a single objective, such as the pathogen's attack rate, while the mitigation of epidemics is a problem that inherently covers distinct and possibly conflicting criteria (i.a., prevalence, mortality, morbidity, cost). Since somehow aggregating these criteria into a single metric comes with many challenges, we have argued in Chapter 2 for an explicitly multi-objective approach.

In this Chapter, we investigate the use of MORL to learn a set of solutions that approximate the Pareto front of multi-objective mitigation strategies. We consider the first wave of the Belgian COVID-19 epidemic, which was mitigated by a hard lockdown (Willem et al., 2021). When the incidence of confirmed cases were steadily dropping, epidemiological experts were asked to investigate strategies to exit from the stringent lockdown which was imposed.

Here, we consider the epidemiological model developed by (Abrams et al., 2021) that was constructed to describe the Belgian COVID-19 epidemic, and was fitted to hospitalization incidence data and serial sero-prevalence data. This model constitutes a stochastic discrete-time age-structured compartmental model that simulates mitigation strategies by varying social distancing parameters concerning school, work and leisure contacts.

Based on this model, we contribute MOBelCov, a novel mutli-objective epidemiological environment, in the form of a multi-objective Markov decision process (Roijers et al., 2013). MOBelCov encapsulates the epidemiological model developed by (Abrams et al., 2021) to implement state transitions, with an action space that combines a proportional reduction of school, work and leisure contacts at each time step. Furthermore, it defines a reward function based on two objectives: the attack rate (i.e., proportion of the population affected by the pathogen) and the social burden that is induced by the mitigation measures.

To learn and explore the trade-offs between the attack rate and social burden we apply PCN, described in Section 5.3. As PCN is an algorithm designed for discrete action-spaces, we extend it towards continuous action-spaces to accommodate MOBelCov's action-space. With this continuous action variant of PCN, we explore the Pareto front of multi-objective

COVID-19 mitigation policies.

By evaluating the solution set of mitigation policies returned by PCN, we observe that PCN minimizes the social burden in scenarios where hospitalization rates are sufficiently low. Therefore, in this work we illustrate that multi-objective reinforcement learning can be utilized to provide important insights surrounding the trade-offs between complex mitigation polices in real-world epidemiological models.

6.2 COVID-19 model and the MOBelCov MOMDP

6.2.1 Stochastic compartment model for SARS-CoV-2

As an environment to evaluate non-pharmaceutical interventions, we consider the adapted version of the SEIR compartmental model presented by Abrams et al., that was used to investigate exit strategies in Belgium after the first epidemic wave of SARS-CoV-2 (Abrams et al., 2021).

In a SEIR model, the population is separated into 4 different compartments. Members of the population are susceptible to infection when they are in compartment $\mathbf{S}$[1]. If an individual comes into contact with an infections individual then they move to the exposed compartment, $\mathbf{E}$, at a time specific rate, $\boldsymbol{\lambda}(t)$. After a period of time an exposed individual becomes infectious, where they move to the infected compartment, $\mathbf{I}$. Finally, infected people recover, at which point they move to the recovered compartment, $\mathbf{R}$.

However, to make efficient mitigation strategies, particularly ones that can deal with intensive care capacity or implement quarantines of symptomatic people, it is essential to extend this model with additional relevant compartments. To this end, the compartment model we use captures the different stages of disease spread and history that are associated with SARS-CoV-2 (i.e., pre-symptomatic, asymptomatic, symptomatic with mild symptoms and symptomatic with severe symptoms) and to represent the stages associated with severe disease, i.e., hospitalization, admission to the intensive care unit (ICU) and death. Thus, after a person becomes infected, they move to a new pre-symptomatic compartment, $\mathbf{I}_{presym}$, at rate γ. Once infected, individuals develop mild symptoms, $\mathbf{I}_{mild}$, with probability $1 - \mathbf{p}$ or do not develop any symptoms, $\mathbf{I}_{asym}$, with probability $\mathbf{p}$, where asymptomatic individuals recover, $\mathbf{R}$, at rate δ_1. Individuals who experience symptoms can suffer from a mild infection, $\mathbf{I}_{mild}$, and recover at rate $\boldsymbol{\delta}_2$, or they suffer from a more severe infection, $\mathbf{I}_{sev}$, at a rate $\boldsymbol{\psi}$. Individuals with a severe infection are then transferred to a hospital for treatment, $\mathbf{I}_{hosp}$ with probability ϕ_1. However, some individuals become critically ill and are transferred directly to the ICU with probability $1 - \psi_1$. Individuals in the hospital, $\mathbf{I}_{hosp}$ and $\mathbf{I}_{ICU}$, recover at rate $\boldsymbol{\delta}_3$ or $\boldsymbol{\delta}_4$ and die at rate $\boldsymbol{\tau}_3$ and $\boldsymbol{\tau}_4$ respectively. Figure 6.1 outlines the compartmental model defined by Abrams et al. (Abrams et al., 2021).

The flow rates depicted in Figure 6.1 are defined by a set of ordinary differential equations,

[1]Boldface vector notation is used to denote the multiple age groups for each compartment.

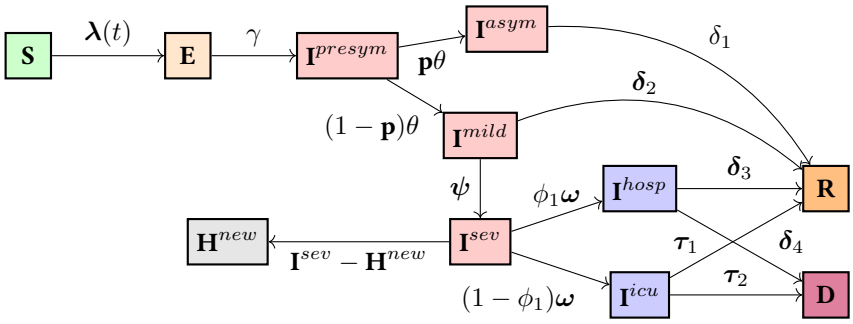


Figure 6.1: Schematic diagram of the compartmental model for SARS-CoV-2 presented by (Abrams et al., 2021) which is used to derive the MOMDP.

which are outlined as follows:

$$
\begin{aligned}
\frac{d\mathbf{S}(t)}{dt} &= -\lambda(t)(\mathbf{S})(t), \\
\frac{d\mathbf{E}(t)}{dt} &= \lambda(t)(\mathbf{S})(t) - \gamma\mathbf{E}(t), \\
\frac{d\mathbf{I}_{presym}(t)}{dt} &= \gamma\mathbf{E}(t) - \theta\mathbf{I}_{presym}(t), \\
\frac{d\mathbf{I}_{asym}(t)}{dt} &= \theta p\mathbf{I}_{presym}(t) - \delta_1\mathbf{I}_{asym}(t), \\
\frac{d\mathbf{I}_{mild}(t)}{dt} &= \theta(1-p)\mathbf{I}_{presym}(t) - \{\psi + \omega_2\}\mathbf{I}_{mild}(t), \\
\frac{d\mathbf{I}_{sev}(t)}{dt} &= \psi\mathbf{I}_{mild}(t) - \omega\mathbf{I}_{sev}(t), \\
\frac{d\mathbf{I}_{hosp}(t)}{dt} &= \phi_1\omega\mathbf{I}_{sev}(t) - \{\delta_3 + \tau_1\}\mathbf{I}_{hosp}(t), \\
\frac{d\mathbf{I}_{ICU}(t)}{dt} &= (1-\phi_1)\omega\mathbf{I}_{sev}(t) - \{\delta_4 + \tau_2\}\mathbf{I}_{ICU}(t), \\
\frac{d\mathbf{D}(t)}{dt} &= \tau_1\mathbf{I}_{hosp}(t) - \tau_2\mathbf{I}_{ICU}(t), \\
\frac{d\mathbf{R}(t)}{dt} &= \delta_1\mathbf{I}_{asym}(t) + \delta_2\mathbf{I}_{mild}(t) + \delta_3\mathbf{I}_{hosp}(t) + \delta_4\mathbf{I}_{ICU}(t)
\end{aligned}
$$

In this set of ordinary differential equations, each state variable represents a vector over all age groups for a particular compartment at time t. For example, $\mathbf{S} = (S_1(t), S_2(t), ..., S_K(t))^T$ is the vector representing the susceptible members of the population of each age group k at time t. Infection dynamics are governed by an age-specific force of infection λ:

$$
\lambda(k,t) = \sum_{k'=1}^{K} \beta(k,k')I_{k'}(t), \tag{6.1}
$$

where K is the total number of age groups, and $\beta(k,k')$ is the time-invariant transmission

rate that encodes the average per capita rate at which an infectious individual in age group k makes an effective contact with a susceptible individual in age group k', per unit of time.

As we consider an age-structured population, we consider this extended SEIR structure for $K = 10$ age groups, i.e., $[0-10), [10-20), [20-30), [30-40), [40-50), [50-60), [60-70), [70-80), [80-90), [90, 100+)$. Contacts of the different age-groups, which impact the propagation rate of the epidemic, are modeled using social contact matrices. We define a social contact matrix for 6 different social environments: $C_{\text{home}}, C_{\text{work}}, C_{\text{transport}}, C_{\text{school}}, C_{\text{leisure}}, C_{\text{other}}$, for the home, work, transport, school, leisure, other environments respectively. The social contact matrix across all social environments is defined as:

$$C = C_{\text{home}} + C_{\text{work}} + C_{\text{transport}} + C_{\text{school}} + C_{\text{leisure}} + C_{\text{other}} \tag{6.2}$$

Under the social contact hypothesis (Wallinga et al., 2006), we have that:

$$\beta(k, k') = q \cdot C(k, k'), \tag{6.3}$$

where q is a proportionality factor.

Following (Abrams et al., 2021), we rely on distinct social contact matrices for symptomatic and asymptomatic individuals, respectively C_s and C_a. Therefore, we define the transmission rates for both symptomatic and asymptomatic individuals as follows:

$$\boldsymbol{\beta}_s(k, k) = q_s \cdot C_s(k, k'), \tag{6.4}$$

and

$$\boldsymbol{\beta}_a(k, k') = q_a \cdot C_a(k, k'). \tag{6.5}$$

The age-dependent force of infection can be defined as follows:

$$\boldsymbol{\lambda}(t) = \boldsymbol{\beta}_a \times \{\mathbf{I}_{presym}(t) + \mathbf{I}_{asym}(t)\} + \boldsymbol{\beta}_s \times \{\mathbf{I}_{mild}(t) + \mathbf{I}_{sev}(t)\}, \tag{6.6}$$

where $\boldsymbol{\lambda}(t) = (\lambda(1, t), \lambda(2, t), ..., \lambda(K, t))$. For all further information about the different compartments and parameters we refer the reader to the work of (Abrams et al., 2021).

Variability in social contact behavior, disease transmission, recovery and mortality impacts the epidemic outcome and is subject to chance. To evaluate intervention strategies that modulate infectious disease transmission and prevention, the use of stochastic epidemiological models is warranted (Abrams et al., 2021). Moreover, the effect of stochasticity is most pronounced when the number of infectious individuals is small or variability is high, for example studying the initial growth of an epidemic (Britton & Lindenstrand, 2009), or when implementing deconfinement strategies after strict lock-downs.

By formulating the set of differential equations defined above, as a chain-binomial (see Appendix B.1), we can obtain stochastic trajectories from this model (Bailey, 1975). A chain-binomial model assumes a stochastic model where infected individuals are generated by some underlying probability distribution. For this stochastic model we consider a time interval $(t, t + h]$, where h is defined as the length between two consecutive time points. In this work we set $h = \frac{1}{240}$.

6.2.2 Interventions strategies

To model different types of non-pharmaceutical interventions, we follow the contact reduction scheme presented by (Abrams et al., 2021). Firstly, in order to consider distinct exit scenarios, we modulate the SCMs to reflect a contact reduction in a particular age group. We consider a contact reduction function that imposes a proportional reduction of work (including transport) p_w, school p_s and leisure p_l contacts, which is implemented as a linear combination of SCMs:

$$\hat{C}(p_w, p_s, p_l) = C_{\text{home}} + p_w(C_{\text{work}} + C_{\text{transport}}) + p_s C_{\text{school}} + p_l(C_{\text{leisure}} + C_{\text{other}}) \quad (6.7)$$

We denote $\hat{C}_t$ the social contact matrix at timestep t, resulting from the reduction function $\hat{C}$. Secondly, we assume that compliance to the interventions is gradual and model this using a logistic compliance function (see details in Appendix B.1.1).

We can thus simulate the lockdown that occurred during the first wave of the pandemic by fixing p_w, p_s and p_l for the concerned time-period. Figure 6.2 shows the progress of the epidemic during lockdown, by setting $p_w = 0.2, p_s = 0, p_l = 0.1$.

6.2.3 A MOMDP for the compartment model

In order to apply multi-objective reinforcement learning, we construct the MOBelCov MOMDP based on the epidemiological model we introduced. Since a MOMDP is a tuple, $\mathfrak{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \boldsymbol{\mathcal{R}}, n \rangle$, we define its components $\mathcal{S}, \mathcal{A}, \mathcal{T}, \boldsymbol{\mathcal{R}}$.

Action-space: Our actions concern the installment of a SCM with a particular reduction resulting from the reduction function $\hat{C}$ (see Sec. 6.2.2). To this end, we use the proportional reduction parameters p_w, p_s, p_l defined in Sec. 6.2.2. Thus, each $a \in \mathcal{A}$ is a 3-dimensional continuous vector in $[0, 1]^3$ (i.e., $a = [p_w, p_s, p_l]$) which impacts the SCM according to Equation 6.7.

Transition function: The model defined by (Abrams et al., 2021) utilizes a model transition probability M (see Appendix B.1 for details on M), that progresses the epidemiological model in one timestep based on the currently installed SCM $\hat{C}(p_w, p_s, p_l)$. We use this function as the transition function in MOBelCov.

In a classical MDP, executing an action a_t in any state s_t leads to a next state s_{t+1} according to the transition function $\mathcal{T}$. At every timestep t, the policy is free to choose the action to perform. In our case, this potentially results in a different restriction $[p_w, p_s, p_l]$ every week. However, we argue that in the context of mitigation policies, consistency is key, and policies that impose changes too frequently will be hard to follow.

In order to learn realistic and consistent mitigation policies, we incorporate a *budget* on the number of times a policy can change its actions until the terminal state of the MOMDP. Concretely, when the action changes, i.e., if the social restriction proposed by the policy is different from the current one in place, we reduce the budget by one. We only allow action

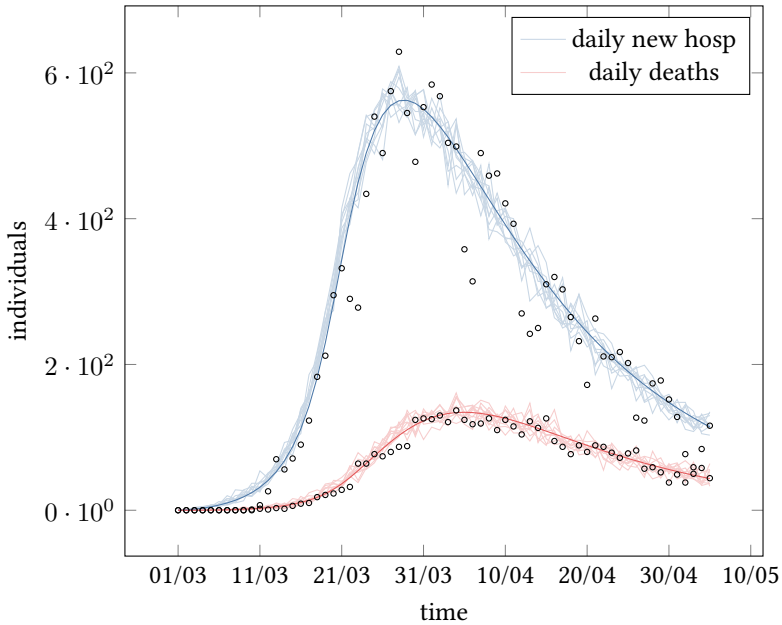


Figure 6.2: Simulation of the modified SEIR compartment model from the start of the epidemic until the end of the impose lockdown, by setting the SCM using Equation 6.7 with $p_w = 0.2, p_s = 0, p_l = 0.1$ during the lockdown period. We plot 10 executions of the lockdown policy and observe a variation in the evolution of the epidemic due to the environment stochasticity. In bold, we show a policy execution on the deterministic variant of the environment. Finally, the scatter-points correspond to reported hospitalizations and deaths related to COVID-19 in Belgium during that time-period (see Appendix **??**).

changes as long as there is budget left. We note that we can simulate a no-limit budget setting by setting the budget to the length of the episode.

Finally, for each timestep t, our transition function $\mathcal{T}$ uses the model transition probability M to simulate the model for one week, using $\hat{C}$ obtained from a_t.

State-space: The state of the MOMDP concerns a 3-tuple. The first element, s_m, directly corresponds to the aggregation of the state variables in the epidemiological model, i.e., a tuple,

$$\{S_k, E_k, I_k^{presym}, I_k^{asym}, I_k^{mild}, I_k^{sev}, I_k^{hosp}, I_k^{ICU}, H_k^{new}, D_k, R_k\},$$

for each age group $k \in \{1, \ldots, K\}$, where D encodes the members of the population who become deceased, and H^{new} explicitly keeps track of the number of newly hospitalized individuals.

We parameterize the epidemiological model using the mean of the posteriors as specified by Abrams et al. (2021). The population size for each of the considered age groups was taken from the Belgian statistical agency STATBEL[2]. To initialize the model, we used the

[2]https://statbel.fgov.be/nl/themas/bevolking/structuur-van-de-bevolking#figures

number of confirmed cases until 13 March 2020 (Abrams et al., 2021), as reported by the Belgian agency for public health Sciensano[3].

The second element of the tuple consists of the social contact matrix $\hat{C}_t$ that is currently in place. The reason to incorporate it in the state-space is two-fold. First, Abrams et al. (2021) define a compliance function, simulating the time people need to get used to the new rules set in place. As such, during the simulated week, there is a gradual shift from the current $\hat{C}_t$ to the new social contact matrix, $\hat{C}_{t+1}$. As such, we need to maintain the current $\hat{C}_t$, to maintain a Markovian environment.

Secondly, we require the current $\hat{C}_t$ to determine whether the action changes the social restrictions in place, and thus consume part of the budget.

The third element of the tuple concerns of the budget b. We note that we incorporate a distinct budget per action-dimension, so p_w, p_s and p_l each have their own budget, rendering b a vector $b = [b_w, b_s, b_l]$. As such, it is possible that, at timestep t, the budget for one of the proportional reductions is reduced but not the others.

Therefore, we define a state in MOBelCov as follows:

$$s = s_m \cup \hat{C} \cup b \tag{6.8}$$

Reward function: We define a vectorial reward function which considers multiple objectives: attack rate (i.e., infections, hospitalizations) and the social burden imposed by the interventions on the population.

The attack rate in terms of infections is defined as the difference in susceptibles from the current state and next state (P. J. K. Libin et al., 2021). Since this is a cost that needs to be minimized, we defined the corresponding reward function as the negative attack rate:

$$\mathcal{R}_{\text{ARI}}(s, a, s') = -(\sum_{k=1}^{K} S_k(s) - \sum_{k=1}^{K} S_k(s')). \tag{6.9}$$

The reward function to reduce the attack rate in terms of hospitalizations is defined as the negative number of new hospitalizations:

$$\mathcal{R}_{\text{ARH}}(s, a, s') = -\sum_{k=1}^{K} H_k^{\text{new}}(s). \tag{6.10}$$

Finally, we use the missed contacts resulting from the intervention measures as a proxy for societal burden. To quantify missed contacts, we consider the original social contact matrix C and the installed social contact matrix $\hat{C}_t$ to compute the difference $\hat{C}_t - C$. The resulting difference matrix quantifies the average frequency of contacts missed. To determine missed contacts for the entire population, we apply the difference matrix to the population sizes of the respective age groups that are currently uninfected (i.e., susceptible

[3] https://epistat.wiv-isp.be/covid/

and recovered individuals). Therefore, we define the social burden reward function $\mathcal{R}_{\text{SB}}$, as follows:

$$\mathcal{R}_{\text{SB}}(s, a, s') = \sum_{i=1}^{K}\sum_{j=1}^{K}(\hat{C} - C)_{ij} S_i(s) S_j(s) + \sum_{i=1}^{K}\sum_{j=1}^{K}(\hat{C} - C)_{ij} R_i(s) R_j(s), \quad (6.11)$$

where $S_k(s)$ represents the number of susceptible individuals in age group k in state s and R_k represents the number of recovered individuals in age group k in state s.

In Section 6.4, we optimize PCN on two different variants for the multi-objective reward function: $[\mathcal{R}_{\text{ARH}}, \mathcal{R}_{\text{SB}}]$ and $[\mathcal{R}_{\text{ARI}}, \mathcal{R}_{\text{SB}}]$, to study the impact of these distinct attack rate quantities.

6.3 Pareto Conditioned Networks for MOBelCov

In Section 5.3, we propose Pareto Conditioned Networks (PCN), an algorithm to learn the Pareto front and its associated set of Pareto-efficient policies. Since our aim is to learn all possible non-dominated trade-offs between hospitalizations and social burden, we apply PCN on our proposed MOBelCov environment.

However, PCN learns on environments with discrete action-spaces, while the action-space of MOBelCov is continuous. Moreover, since PCN learns by imitating trajectories, it is not designed to learn on MOMDPs with a stochastic transition function.

In this Section, we extend PCN towards continuous action-spaces. Additionally, we propose a modified training scheme to cope with the stochasticity of the MOBelCov environment.

6.3.1 Training the network for continuous actions

PCN trains the network as a classification problem, where each class represents a different action. Transitions $x = \langle s_t, h_t, \boldsymbol{R}_t \rangle, y = a_t$ are sampled from the dataset, and the ground-truth output y is compared with the predicted output $\hat{y} = \pi(s_t, h_t, \boldsymbol{R}_t)$. The predictor (i.e., the policy) is then updated using the cross-entropy loss function (see Section 5.3.2).

To cope with the continuous action-space setting, we change the output of the neural network such that there is a single output value for each dimension of the action-space. Since the actions should be bound in the domain of possible actions ($[0, 1]$ in the case of MOBelCov, see Section 6.2.3), we apply a tanh non-linearity function on each output. As such, we have a regression problem instead of a classification problem, as the labeled dataset now uses continuous labels $y = a_t$ instead of categories. We thus use a Mean Squared Error (MSE) loss to update the policy:

$$MSE = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} (\hat{y}_a - y_a)^2 \quad (6.12)$$

Since learning the full set of Pareto-efficient policies Π^* requires that the policies $\pi^* \in \Pi^*$ are deterministic stationary policies (Roijers et al., 2013), we use the output $\hat{\mathbf{y}}$ as action at execution time. However, PCN improves its policy through exploration, by continuously updating its dataset with better trajectories. Thus, at training time, we use a stochastic policy by adding random noise to the action (Lillicrap et al., 2015):

$$a_t = \pi(a_t, h_t, \boldsymbol{R}_t) + \eta n \text{ with } n \sim \mathcal{N}, \tag{6.13}$$

where $\mathcal{N}$ is the standard Normal distribution and η is a hyper-parameter defining the magnitude of noise to be added.

6.3.2 Coping with stochastic transitions

PCN trains its policy on a dataset that is collected by executing trajectories. It assumes that reenacting a transition from the dataset leads to the same episodic return. When the transition function $\mathcal{T}$ of the MOMDP is deterministic, the whole trajectory can be faithfully reenacted, which guarantees the same return. Combined with the fact that PCN's policy is deterministic at execution time, conditioning the policy on a target episodic return is equivalent to conditioning it on the Value $\boldsymbol{V}$.

However, when $\mathcal{T}$ is stochastic this can no longer be guaranteed. To mitigate this, we add small random noise to $\boldsymbol{R}_t$ when performing gradient descent, which reduces the risk of overfitting (Zur et al., 2009). Moreover, while the MOBelCov model is stochastic, the variation is entirely due to the sampling of the binomial distributions in the binomial-chain. While this variation accumulates over time, the time window we consider for each timestep (i.e., one week) is short enough that the accumulation stays limited. Thus, the possible next-states resulting from a state-action pair are similar to each other. This allows PCN to compensate if $\boldsymbol{r}_t = \mathcal{R}(s_t, a_t, s_{t+1})$ is worse than expected.

Although we use a stochastic model to cope with the uncertainty of the outcome of the outbreak, it is possible to deterministically evaluate the set of ordinary differential equations that define the model. We assess the validity of our approach by executing PCN on this deterministic variant, and observing similar performance as with the MOBelCov model. We show the results in Appendix B.2.1.

6.4 Experiments

Our goal is to use PCN to learn deconfinment strategies in the MOBelCov environment. We aim to learn policies that balance between the epidemiological objective of minimising the attack rate (i.e., $\mathcal{R}_{\text{ARH}}$ for hospitalization and $\mathcal{R}_{\text{ARI}}$ for infection) and the social burden (i.e., $\mathcal{R}_{\text{SB}}$) experienced by the population population due to the implement mitigation measures. To this end, we consider two cases for the vectorial reward functions $[\mathcal{R}_{\text{ARH}}, \mathcal{R}_{\text{SB}}]$ and $[\mathcal{R}_{\text{ARI}}, \mathcal{R}_{\text{SB}}]$, to learn and analyse policies under different targets with respect to the considered attack rate.

To conduct this analysis, we apply our extension of PCN for continuous action-spaces on the MOBelCov model. As explained in Section 6.3.2, we extend PCN for environments with stochastic transitions.

As in the work of Abrams et al. (2021), the simulation starts on the 1st of March 2020, by seeding a number of infections in the population. Two weeks later, on the 14th of March, the Belgian government initiated a full lockdown. This is implemented by fixing the actions p_w, p_s, p_l to $0.2, 0, 0.1$ respectively (see Figure 6.2). This lockdown ended on the 4th of May 2020, at which point the government decided on a multi-phase exit strategy to incrementally reduce teleworking, reopen schools and allow leisure activities, such as the re-opening of bars and the cultural sector. It is from this day onward that PCN aims to learn policies for diverse exit strategies, compromising between the total number of daily hospitalizations and the total number of contacts lost as a proxy for social burden. The simulation lasts throughout the summer school holidays, from 01/07/2020 to 31/08/2020. Schools are closed during the school holidays, which is simulated by setting $p_s = 0$, regardless of the corresponding value outputted by the policy, i.e., during periods of school closure p_s is ignored.

We draw the analogy with the multi-phase exit strategy established by the Belgian government and the restriction on the number of action-changes imposed by the MOBelCov's budget b. Indeed, on the 11th of May 2020, exactly one week after the end of the lockdown, stores and certain companies were allowed to reopen, under strict conditions. This corresponds to altering p_w in our MOMDP. One week later, on 18th of May, primary and secondary schools reopened for limited sized class-groups, and the cultural sector reopened partially. This is equivalent to increasing p_s and p_l. Further changes of restrictions occured on the 8th of June, 1st, 9th and 25th of July. Thus, we argue that, with a limited budget, we can achieve realistic policies. In our experiments, we consider budgets of 2 to 5, as these closely relate to the number of changes that occured until the end of the summer holidays of 2020. As an upper bound, we also consider a no-limit budget setting.

To evaluate the quality of the policies learned by PCN, we compare PCN to a baseline. This baseline consists of a set of 100 fixed policies, that iterate over all the possible social restriction levels, with values ranging between 0 and 1. Concretely, each policy uses a fixed proportional reduction $p_w = p_l = p_s = u, u \sim \mathcal{U}(0, 1)$ throughout the episode. In other words, the fixed policies directly operate in a fine-grained manner on the whole contact reduction function $\hat{C}$. This allows us to obtain a strong baseline for potential exit strategies over the objective space. We note that while such fixed policies are a feasible approach, they do not scale well in terms of action and objective spaces and they will not be able to provide an adaptive restriction level, which is our aim using PCN.

All experiments are averaged over 10 runs. The hyper-parameters and the neural network architecture can be found in Appendix B.2.3 and Appendix B.2.4, respectively.

6.4.1 Learned coverage set

We learn a coverage set (see Figure 6.3) that ranges from imposing minimal restrictions to enforcing many restrictions. In Figure 6.3, we display on the right the coverage set of the best-performing run in terms of hypervolume, for each budget setting. On the left, we show an interpolated average of the coverage sets learned by the different runs.

Regardless of the imposed budget, we notice that the coverage sets discovered by PCN almost completely dominate the coverage set of the baseline, demonstrating that there are better alternatives to the fixed policies. This is most evident in the compromising policies,

where one has to carefully choose when to remove social restrictions while at the same time minimizing the impact on daily new hospitalizations. In these scenarios, PCN learns policies that drastically reduce the total number of new hospitalizations (e.g., more than 20000) for the same social burden. We analyse the executions of such policies in Figure 6.4 (middle plot), that shows a flattened hospitalization curve, with a gradual increase of social freedom during the school holidays such that the curve of the epidemic is flattened and gradually decreases over time.

Interestingly, we notice that the most restrictive policy (i.e., the one that prioritizes hospitalizations over social burden, see Figure 6.4, bottom plot) still starts to gradually increase p_w and p_l from the end of July onward. This is because by then, the epidemic has mostly faded out, and it is safe to reduce social restrictions. The timing of this reduction is important as reducing restrictions too soon can lead to a new wave. PCN learns the impact of its decisions over time, and correctly infers the timing at which restrictions can be safely lifted.

Finally, the top plot shows that, without imposing social restrictions, the number of hospitalizations peaks on the 15th of June. By the beginning of July, the epidemic has spread out over the majority of the population, and the number of admissions at the hospital has been reduced to a fraction of the number of hospitalizations at the peak. Thus, without social restrictions, we do not take advantage of the natural decrease of social contacts due to school holidays, as a significant proportion of the population has already been infected before the start of the holidays.

6.4.2 Impact of budgets on the coverage set

Figure 6.3 demonstrates that the budget impacts the learnt coverage set. In general, an increase of budget is associated with an increasingly better coverage set, as policies learned using a higher budget dominate the ones learned with a lower budget. This is to be expected, as a higher budget gives the agent more freedom to change its actions as the epidemic progresses.

Moreover, we observe that the difference is concentrated around the less restrictive policies in terms of social burden. We postulate that this region contains the most complex policies, as these try to maintain as much social freedom as possible, while containing the number of hospitalizations. In these cases, the timing of the actions coincide with the timing and duration of the peak of the epidemic, and a higher budget allows for more fine-grained control to manage this timing. To confirm this, we select, for each budget setting, the solution where the difference in performance is most noticeable, corresponding to the solutions with a total number of hospitalizations around 80000 (which is in the middle of the range of possible hospitalizations, as can be seen in Figure 6.3). We plot the execution of the corresponding policies in Figure 6.5 and analyse their impact in terms of social burden. First, we observe that the lower-budget policies are unable to reduce the social restrictions past the peak of the epidemic. In contrast, the setting with no budget restrictions meticulously controls the restrictions as the epidemic progresses, completely removing restrictions by the end of the wave. Second, we note that the policy with a budget of 5 resembles the execution of the one without restrictions. However, due to its budget, the policy is unable to progressively reduce the restrictions and instead resorts to a halfway

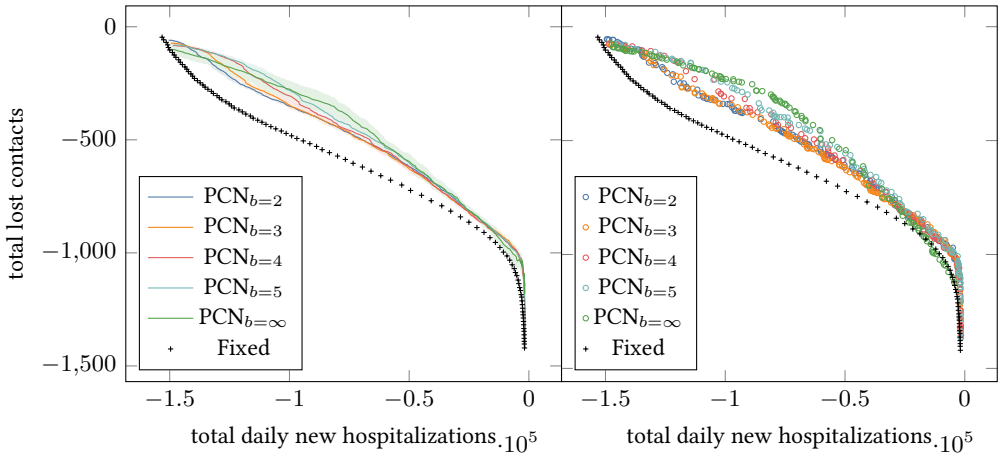


Figure 6.3: The Pareto front of policies discovered by PCN using MOBelCov, showing the different compromises between the number of hospitalizations and the number of lost contacts. On the right, we show, for each budget setting (colored, subscript indicates budget) the coverage set learned by the best performing run. On the left, we show an interpolated average of the coverage sets learned by the different runs, with the shaded regions corresponding to the standard deviation. For comparison, the basline is displayed on both plots (in black). As the budget increases, so does the size of the coverage set learnt by PCN. Changes are most noticeable in the less restrictive trade-offs in terms of social burden.

compromise. Compared to this specific region of the coverage set, the difference in performance between different budget settings seems marginal around the extremas. At the extremas, the policies are less complex (e.g., business-as-usual, resulting in the same action executed throughout the episode) and are thus less impacted by the budget restrictions.

Finally, we observe that, while the extremas deliver similar trade-offs for any of the chosen budgets, these trade-offs differ for the setting without budget restrictions. Indeed, in this setting, PCN does not learn the most extreme policies with respect to restrictions, even though there are no constraints on the action-set. As explained in Section 5.3.3, PCN searches for increasingly better solutions using a stochastic policy. Thus, at every timestep, the action can change compared to the previous one. Continuously outputting the same action (e.g., no social restrictions) becomes a complicated task. In comparison, for the settings with a limited budget, the action stays the same as the previous one once the budget has been spent. As such, it is easier to learn the most extreme policies. Thus, in the specific case where we have an unlimited budget, the freedom of action actually hinders PCN's search, for certain regions of the reward-space.

6.4.3 R_{ARH} versus on R_{ARI}

Next, we assess the difference in coverage sets when optimizing on $\mathcal{R}_{\text{ARH}}$ versus $\mathcal{R}_{\text{ARI}}$. Although these reward functions have a different scale (there are more infected persons than hospitalised ones), our experiments show that infections and hospitalizations are tightly

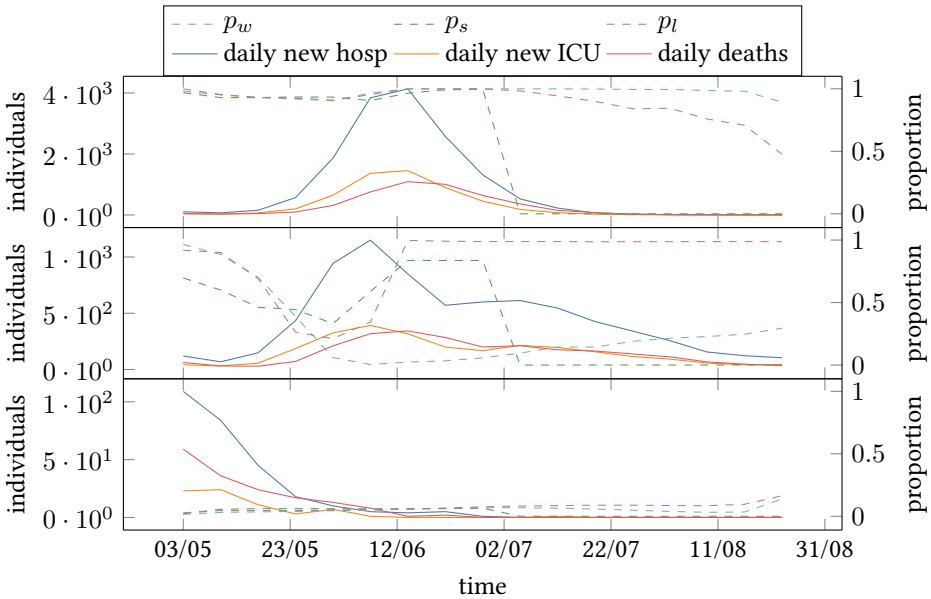


Figure 6.4: Selection of policies learned by $\text{PCN}_{b=5}$, from most restrictive in terms of social burden (top) to least restrictive (bottom). The x-axis represents the time, starting from the end of the lockdown, on the 4th of May, until the end of the school holidays, on the 1st of September. Since the lockdown is simulated before the start of the exit strategy, the start-state differs for each episode (i.e., the hospital already contains infected individuals). The left y-axis represents the number of individuals affected by the epidemic. The full-lined plots represent the number of individuals admitted into the hospital, ICU and deceased between the last timestep and the current one. The plot showing the newly deceased individuals closely relates to the ICU admissions. The right y-axis represents the proportional reduction in effect, with 1 meaning a business-as-usual policy, and 0 meaning a complete suppression of social contacts. The dotted-lined plots represent the proportional reductions for the work, leasure and school environments. We note that the school reduction automatically goes to 0 at the start of the school holidays.

correlated. This is expected, as during the initial phase of the epidemic, limited immunity was present in the population (i.e., limited natural immunity and no vaccines), which induces a tight coupling between infection and hospitalization cases. This is confirmed in Table 6.1. In this table, we show the different performance metrics (hypervolume, I_ε, $I_{\varepsilon-mean}$) with respect to the objectives $[\mathcal{R}_{\text{ARH}}, \mathcal{R}_{\text{SB}}]$. The table is split in two parts. The left-side shows the different performance metrics, for PCN using $[\mathcal{R}_{\text{ARH}}, \mathcal{R}_{\text{SB}}]$ as optimization criterions. The right-side shows the same performance metrics, but with PCN using $[\mathcal{R}_{\text{ARI}}, \mathcal{R}_{\text{SB}}]$ as optimization criterions.

Even with the $\mathcal{R}_{\text{ARI}}$, the increased budget shows an increase in hypervolume in terms of hospitalizations. Moreover, those hypervolumes are close to the ones trained on $\mathcal{R}_{\text{ARH}}$, indicating that their coverage sets are similar. However, regardless of the imposed budget, the hypervolumes are slightly worse. This is to be expected, since those experiments are

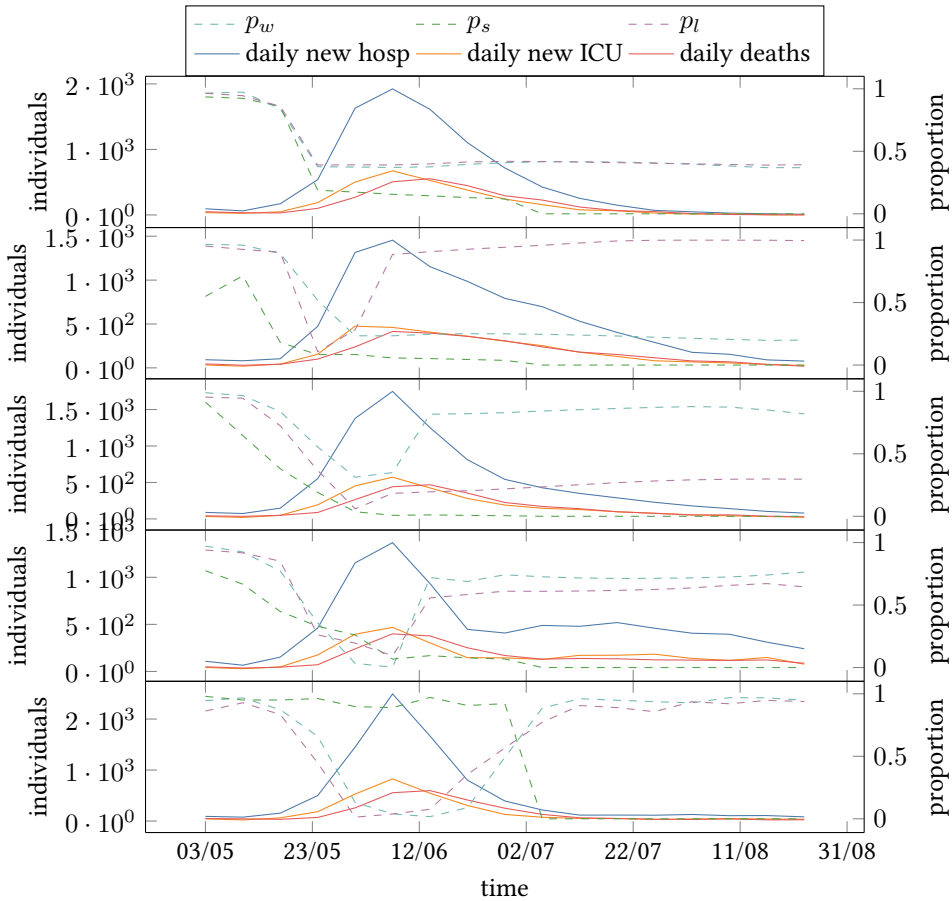


Figure 6.5: Execution of the policies attaining a number of hospitalizations around 80000, for different budgets. From top to bottom we display the policy executions with budget 2,3,4,5 and no-limit, respectively. We notice that the lower-budget policies are unable to reduce the social restrictions past the peak. The setting without budget restrictions finely controls the restrictions as the epidemic progresses, completely removing restrictions by the end of the wave. Finally, there is no consensus on which social environment to restrict most: certain policies provide similar restrictions for p_w and p_l, while others impose harsher restrictions on one social environment than the other.

not directly optimized on $\mathcal{R}_{\text{ARH}}$. We draw a similar conclusion for I_ε: for budgets 2, 3 and 5, the difference between the worst-performing policy for the $\mathcal{R}_{\text{ARI}}$ variant and the $\mathcal{R}_{\text{ARH}}$ is less than 0.01, indicating less than 1% difference in return values between the two variants when comparing their worst-performing policy. As an exception, we notice that PCN without budget restrictions results in better performance across every metric for the $\mathcal{R}_{\text{ARI}}$ variant. Still, due to the high standard deviation of the unlimited budget, $\mathcal{R}_{\text{ARH}}$ setting, we do not believe this difference is meaningful. Thus, we could optimize on the attack rate of hospitalizations with $\mathcal{R}_{\text{ARI}}$. As there is a 2 week delay for hospitalizations,

	$[\mathcal{R}_{\text{ARH}}, \mathcal{R}_{\text{SB}}]$			$[\mathcal{R}_{\text{ARI}}, \mathcal{R}_{\text{SB}}]$		
	Hypervolume	I_ε	$I_{\varepsilon-mean}$	Hypervolume	I_ε	$I_{\varepsilon-mean}$
$\text{PCN}_{b=2}$	158.370 ± 0.811	0.080 ± 0.011	0.033 ± 0.002	157.152 ± 1.023	0.087 ± 0.006	0.035 ± 0.002
$\text{PCN}_{b=3}$	158.721 ± 1.439	0.080 ± 0.012	0.032 ± 0.003	158.002 ± 2.081	0.084 ± 0.009	0.034 ± 0.005
$\text{PCN}_{b=4}$	160.642 ± 1.582	0.075 ± 0.007	0.028 ± 0.003	159.315 ± 2.601	0.088 ± 0.018	0.031 ± 0.006
$\text{PCN}_{b=5}$	163.104 ± 2.386	0.070 ± 0.023	0.022 ± 0.005	161.792 ± 2.464	0.075 ± 0.015	0.026 ± 0.005
Fixed	140.479 ± 0.000	0.139 ± 0.000	0.073 ± 0.000	140.479 ± 0.000	0.139 ± 0.000	0.073 ± 0.000
PCN	159.462 ± 7.713	0.264 ± 0.115	0.036 ± 0.020	159.852 ± 2.395	0.171 ± 0.093	0.032 ± 0.006

Table 6.1: Evaluation metrics for the coverage sets comparing hospitalizations with social burden. In general, an increase of budget results in a better coverage set. Training on infections (ARI) still provides a competitive coverage set in terms of hospitalizations. All PCN coverage sets outperform the baseline.

	I_ε	$I_{\varepsilon-mean}$
$\text{PCN}_{b=2}$	0.047 ± 0.020	0.009 ± 0.004
$\text{PCN}_{b=3}$	0.048 ± 0.022	0.007 ± 0.002
$\text{PCN}_{b=4}$	0.064 ± 0.018	0.011 ± 0.003
$\text{PCN}_{b=5}$	0.058 ± 0.011	0.013 ± 0.003
PCN	0.035 ± 0.011	0.008 ± 0.004
Global average	0.050 ± 0.017	0.010 ± 0.004

Table 6.2: Comparing the difference in the desired return provided to PCN and the actual return PCN obtained when executing its policy. We see that, regardless of the setting, the learned policy faithfully receives a return similar to its desired return.

this would facilitate learning policies to react to unexpected changes earlier than using $\mathcal{R}_{\text{ARH}}$, given that a good proxy to the actual number of infections was available (e.g., due to a scale up of PCR testing, as was the case after the first lockdown).

6.4.4 Robustness of policy executions

The dataset of trajectories that PCN is trained on is pruned over time to keep only the most relevant trajectories. The returns of these trajectories are used in Figure 6.3 to visualize the learned coverage set. Each of these returns can be used as desired return for policy execution. We now assess the robustness of the executed policies, by comparing the return obtained after executing the policy with the corresponding target return. For each run, we execute the policy 10 times and compute the I_ε and $I_{\varepsilon-mean}$ metrics with respect to the coverage set learned during the run. We show that the executed policies reliably obtain returns that are similar to the desired return used to condition PCN.

Results are shown in Table 6.2. The I_ε indicators shows that, regardless of the budget, the decision maker will lose at worst a 0.050 normalized return in any of the objectives. On average, it will lose 0.010 normalized returns, i.e., on average, the return obtained by executing a policy will either result in an additional 1441 hospitalizations than expected, or result in 12 additional social contacts lost. Moreover, we emphasize that the learned

coverage set contains the non-dominated returns encountered over the whole training procedure. Since the MOBelCov model is stochastic, for multiple executions of the same policy, the executions kept in the coverage sets are the ones for which the samples from the binomial-chain resulted in a better progression of the epidemic than average. Thus, we expect our policy-executions to be close to the target selected from the coverage set, but not exactly on target. Based on this analysis, we conclude that the policies trained by PCN are robust and produce returns as close as possible from their chosen target.

6.5 Related work

Reinforcement learning (RL) has been used in conjunction with epidemiological models to learn policies to limit the spread of diseases and predict the effects of possible mitigation strategies (P. J. K. Libin et al., 2021, 2018; Probert et al., 2019).

RL and Deep RL have been used extensively as a decision making aid to reduce the spread of COVID-19. For example, to learn effective mitigation strategies (Ohi et al., 2020), to learn efficacy of lockdown and travel restrictions (Kwak et al., 2021) and to limit the influx of asymptomatic travellers (Bastani et al., 2021).

Multi-objective methods have also been deployed to learn optimal strategies to mitigate the spread of COVID-19. Wan et al. (2021) implement a model-based multi-objective policy search method and demonstrate their method on COVID-19 data from China. Given that this method is model-based, a model of the transition function must be learned by sampling from the environment. The method proposed by Wan et al. (2021) only considers a discrete action space which limits the application of their algorithm. Wan et al. (2021) use linear weights to compute a set of Pareto optimal policies. However, methods which use linear weights can only learn policies on the convex-hull of the Pareto front (Vamplew et al., 2008), therefore the full Pareto front cannot be learned. We note that the method proposed by Kompella et al. (2020) considers multiple objectives. However, the objectives are combined using a weighted sum with hand-tuned weights which are determined by the authors. The weighted sum is applied by the reward function and a single objective RL method is used to learn a single optimal policy. In contrast to previous work, our approach makes no assumptions regarding the scalarisation function of the user and is able to discover Pareto fronts of arbitrary shape.

6.6 Discussion

Making decisions on how to mitigate epidemics has important ethical implications with respect to public health and societal burden. In this regard, it is crucial to approach this decision making from a balanced perspective, to which end we argue that multi-objective decision making is essential. In this work, we establish a novel approach, i.e., an expert system, to study multi-faceted policies, and this approach shows great potential to study future epidemic mitigation policies. We are aware of the ethical implications that expert systems have on the decision process and we make the disclaimer that all results based on the expert system that we propose should be carefully interpreted by experts in the field of public health, and in a broader context that encompasses health economics, well-being and

education. We note that the work in this chapter was conducted by a interdisciplinary consortium that includes computer scientists and scientists with a background public health, epidemiology and bio-statistics.

In this work, we focus on the clinical outcomes of intervention strategies and use the reduced contacts as proxy for social burden. This could be extended into more formal health economic evaluations, by designing reward functions that explicitely consider distinct health economic principles. The COVID-19 pandemic demonstrates the broad impact of infectious diseases on sectors other than health care. This stresses the need to capture a societal and thus multi-objective perspective in the decision making process on public health and health care interventions. Our learned policies confirm this, showing that focusing solely on reducing the number of hospitalizations results in taking drastic measures – more than a thousand social interactions lost per person over the span of 4 months – that may have a long-lasting impact on the population.

Although we use an age-structured compartment model, with social contact matrices to model social interactions, this remains a model that evaluates the progression of the pandemic as an aggregated process over the population. Individual-based models could provide more detailed and localized policies, potentially further improving the quality of possible trade-offs, and would provide an interesting avenue for future work. However, due to the computational cost of simulating such models, and the number of interactions required by reinforcement learning in general, this remains a challenging problem, that will require fundamental research to improve the sample efficiency of multi-objective reinforcement learning algorithms.

While we are able to interpret and analyse the obtained policies and their corresponding trade-offs, as we can plot the Pareto front for two objectives, this approach cannot be used for problems with more objectives, which will be necessary to cover reward functions that cover distinct health economic principles. To facilitate this kind of research, new algorithms are necessary to enable reinforcement learning in many-objective contexts and to interprete the learnt policies.

In this work, PCN is able to cope with model stochasticity, as the stochasticity is limited. In settings where the stochasticity is more pronounced, e.g., when designing policies to control the initial outbreak of an epidemic, further methodological extensions are warranted.

Finally, in this work we studied policies that aim to balance social burden and hospitalizations. Yet, the methodology that we propose shows promise to address a wide variety of public health challenges, such as balancing the number of lost schooldays with respect to the attack rate of infections in schools (Torneri et al., 2021), contact tracing effort compared to the impact of such policies (Willem et al., 2021), the impact of antivirals on the epidemic while balancing the likelihood for resistance mutations to emerge (Torneri et al., 2020), and to balance the cost of universal testing and its impact on an emerging epidemic (P. J. Libin et al., 2021).

To conclude, we show that multi-objective reinforcement learning can be used to learn a wide set of high-quality policies on real-world problems, providing the decision maker with insightful and diverse alternatives, and showing the impact of extreme measures. Furthermore, we show that action budgets can act as a regulariser that facilitates learning realistic policies that can be easily conveyed to decision makers.

Chapter 7

Conclusions and future work

Throughout this dissertation, we have emphasized the importance of user-focused optimization. By taking into account the decision maker's preferences, we have incorporated him or her in the optimization process. We have seen that this personalized approach has multiple consequences.

First, combining the different objectives in a utility for optimization is non-trivial, as the utility function can be non-linear. For sequential decision-making, such as reinforcement learning, this non-linearity breaks the sum-of-rewards assumption used in the single-objective case (Section 2.5). Thus, even if we know the utility function a priori, care needs to be taken to actually optimize on this utility. We have shown different methods of incorporating known, non-linear utility functions in the optimization process, either by augmenting the state-space with accrued rewards (Section 3.3), or by optimizing on the different objectives and using the utility function to guide the learner to the preferred trade-off. We have proposed a policy-gradient based method for the second case (Section 3.6,3.4), proven its convergence to a local optimum, and empirically shown that this approach results in better sample-efficiency and lower variance during training compared to using single-objective solvers that directly optimize on the utility (Section 3.9).

Second, when the utility function is not known a priori, we can learn it in conjunction with the optimal trade-off. Thus, we are interleaving two different tasks into one. We have seen that both processes are intertwined, and that, by optimizing both the policy and our estimate over the utility function simultaneously, we can significantly improve our chances of learning the optimal policy (Section 4.4.6). We have proposed a novel algorithm for multi-objective multi-armed bandits that optimizes the interactions with the decision maker, such that it maximizes our chances of improving the policy (Section 4.5). While computationally expensive, this approach improves our chances of learning the optimal policy with a limited number of training steps and interactions, even compared to intertwined approaches with fixed interaction-timings (Section 4.5.5).

Finally, in the case we would like to take into account the decision maker's preferences, but we cannot interact with him or her to gather knowledge about its utility, we can learn all possible Pareto-efficient trade-offs. In contrast to the previously mentioned consequences,

this setting results in needing to learn multiple, different policies instead of the single, preferred policy by the user. Since diverse parts of the state-space can result in different trade-offs, exploration becomes paramount. To tackle this challenge, we have proposed a novel algorithm that uses a single, conditional neural network to encode all the Pareto-efficient policies (Section 5.3). The conditioning is based on the desired trade-off the decision maker wants to achieve. This result in a generalization of the encountered parts of the state-space across the different policies, greatly improving exploration and sample efficiency (Section 6.4). Moreover, this conditioning allows scaling to problems with a high number of objectives (Section 5.5).

Although the consequences of taking a personalized approach w.r.t. the decision maker result in diverse settings, and we need to design setting-specific algorithms, we argue that, as an end-result, taking the multi-objective approach is beneficial for the decision maker, as he or she becomes involved in the optimization process. We demonstrate this on the concrete, real-world use-case of epidemic outbreak mitigation. Our aim is to provide the decision maker with additional insights on the different evolutions of a pandemic (in this case the first wave of COVID-19 in Belgium), using available social restriction policies (such as a lockdown). By learning the full Pareto front of policies, the decision maker can better understand the impact of (partially) closing different social environments on the number of hospitalizations due to COVID-19. For example, we can avoid a full closure of all social environments, as the difference between full closure and an optimized 65% closure is limited in terms of hospitalizations (7% compared to the business-as-usual scenario).

7.1 Future work

The research performed during this dissertation led to various ideas, which we believe could lead to interesting future work. We summarize the different settings for the utility-based perspective in multi-objective reinforcement learning in Table 7.1, including a non-exhaustive but representative list of relevant work. We note that the mentioned works for the knowledge-gathering non-linear utility functions setting for MOMDPs (in gray) are not multi-objective works, but preference-based learning based on trajectories. As they are the closest to the corresponding setting, we include them in the table. We believe that these works can serve as an inspiration to extend our belief distribution over the utility function and query-selection mechanism (Section 4.4.2) from the MOMAB setting to the MOMDP setting.

Moreover, we note that the ESR setting has been generally understudied compared to the SER setting and, to the best of our knowledge, not studied in the context of MOMABs. For the incomplete-knowledge scenario setting, an ESR solution concept has only recently been proposed (Hayes, Verstraeten, et al., 2021), which is why no other work tackles it. Moreover, a large body of work in MORL assumes linear utility functions, for which ESR and SER are equivalent. Additionally, ESR has only recently been studied in the known-utility function setting. We believe the ESR optimization criterion is relevant for many applications, such as energy generation under strict emission constraints (Mannion et al., 2016) or medical treatments (Jalalimanesh et al., 2017; Laber et al., 2014; Lizotte et al., 2010), as for these applications each policy execution is critical, and their utility function can be non-linear. Considering that MOPOMCP keeps a belief distribution over arms

for unknown mean and unknown variance, we could take a distributional approach similar to MOCAC to extend MOPOMCP to the ESR setting, after having incorporated belief distributions for non-linear utility functions. Additionally, we can improve MOCAC by incorporating elements from recent advances in the actor-critic paradigm (Andrychowicz et al., 2021; Haarnoja et al., 2018) and distributional reinforcement learning (on which we expand later).

	MOMABs		MOMDPs	
	ESR	SER	ESR	SER
Full-knowledge				
linear u	(Kagrecha et al., 2023)		(Neil et al., 2018; Peng et al., 2018; Tesauro et al., 2008)	
non-linear u			(Hayes, Reymond, et al., 2021; P. Zhang et al., 2021)MOCAC (Reymond, Hayes, et al., 2023)	(Van Moffaert et al., 2013b)
Knowledge-gathering				
linear u	(Astudillo & Frazier, 2020)			
non-linear u		(Astudillo & Frazier, 2020; Paria et al., 2020; Roijers et al., 2017)MOPOMCP [Section 4.5]		(Christiano et al., 2017; Ibarz et al., 2018; Wirth et al., 2016, 2017)
Incomplete-knowledge				
linear u			(Abels et al., 2019; Alegre et al., 2023; Alegre et al., 2022; Castelletti et al., 2012)	
non-linear u		(Belakaria et al., 2020; Daulton et al., 2020; Yahyaa & Manderick, 2015; R. Zhang & Golovin, 2020)	(Hayes, Verstraeten, et al., 2021)	(Parisi et al., 2014, 2017; Reymond & Nowé, 2019)PCN (Reymond, Eugenio, & Nowè, 2022)

Table 7.1: Overview of all identified settings for the utility-based perspective in multi-objective reinforcement learning. In blue, we mention the contributions exposed in this dissertation for the identified settings. We note that the mentioned works for the knowledge-gathering non-linear utility functions setting for MOMDPs (in gray) are not multi-objective works, but preference-based learning based on trajectories. Moreover, an ESR solution concept for the incomplete-knowledge scenario has only recently been proposed (Hayes, Verstraeten, et al., 2021), which is why no other work tackles this setting.

Concerning the SER optimization criterion, we have derived a proof of convergence of the policy gradient theorem for known non-linear utility functions (Section **??**), which has been used in MOAC, our proposed baseline for the SER setting (Section 3.6.3). We believe our policy gradient for SER can be incorporated in recent single-objective actor-critic methods (Andrychowicz et al., 2021) to design efficient MORL algorithms for known non-linear utility functions under the SER criterion.

Finally, diverse related RL fields share similarities or challenges to MORL. We outline some of them below, and how they could inspire future work.

7.1.1 Conditional reinforcement learning

Conditioning neural networks on values has lead to interesting results in various areas of deep learning. For example, in image generation, conditional GANs (Mirza & Osindero, 2014) allow to generate images, conditioned on desired labels (e.g., dog, cat). Analogously, text-to-image diffusion models condition the generated image on a caption text (Saharia et al., 2022). In the pharmaceutical domain, variational autoencoders have been used to generate molecules conditioned on desired properties (Lim et al., 2018).

In reinforcement learning, this idea led to goal-conditioned reinforcement learning (Andrychowicz et al., 2017; Nasiriany et al., 2019), universal value functions (Schaul et al., 2015), generalized policy improvement (Barreto et al., 2020), dynamic weights (Abels et al., 2019), as well as return conditioned policies such as PCN (Chapter 5). These methods all generalize the single-policy setting in different ways, by e.g. learning a policy or a V-value function conditioned on a goal state, or weights of linear utility functions Additionally, the accrued rewards necessary for optimization under known utility functions (Section 3.2) can also be seen as a form of conditioning.

We believe this idea can be further extended in MORL, through different potential avenues. For example, the dynamic weights setting (Abels et al., 2019) has been limited to linear utility functions, but by using other scalarization methods, e.g. using Chebyshev scalarization (Section 2.4), could be extended to non-linear utility functions. Moreover, we could incorporate weight-conditioned policies into the interactive setting (Chapter 4), by e.g. conditioning on the belief distribution over the utility function, allowing to take into account uncertainty over the optimal policy.

7.1.2 Distributional reinforcement learning

Distributional value networks have been incorporated in DQN in multiple fashions (Bellemare et al., 2017; Dabney et al., 2018; Hessel et al., 2018; D. Yang et al., 2019), greatly improving sample efficiency compared to non-distributional variants, with (Vieillard et al., 2020) resulting in similar performance as model-based approaches (R. Agarwal et al., 2021). It appears that, for the single-objective case, the benefits from using a distribution come from regularizing effects of modelling said distribution, and its role as an auxiliary task in a deep learning context, not really from reinforcement learning principles (Lyle et al., 2019).

Learning return distributions has also been used in a risk-aware setting (Hayes et al., 2023; Morimura et al., 2010), where the distribution can be used to assess the probability of an

action resulting in low-valued returns.

We argue that, in the multi-objective setting, learning such a distribution presents additional advantages, that we could further exploit (Röpke et al., 2023). First, when the utility function is unknown, and diverse parts of the state-space need to be explored, targeted exploration becomes paramount. We believe the distribution could be used to direct the policy towards promising trade-offs, greatly improving the speed at which we expand the learned coverage set. In a similar fashion, distributions that showcase a high uncertainty could indicate that the associated actions have not been performed enough, as their consequences are still uncertain.

Finally, learning return distributions is essential when optimizing on the ESR criterion (Chapter 3), a criterion that has been under-exploited in MORL, but presents many practical uses, e.g., medical treatments (Jalalimanesh et al., 2017), generator scheduling (Mannion et al., 2018), stock investing (Y. Shen et al., 2014). We believe that further extending our work to e.g. parametric multivariate distributions using normalizing flows (Dinh et al., 2017; Hayes et al., 2022), or learn the inverse cumulative distribution function instead of categories, as in (D. Yang et al., 2019), is essential for new ESR-optimized MORL algorithms.

7.1.3 Model-based reinforcement learning

One way to improve sample efficiency in RL is to use a model, either given, or learned (Hafner et al., 2020; Moerland et al., 2023). We can create fictitious interactions through this model, avoiding costly interactions with the real environment. Moreover, these models have also been used to target exploration to parts of the state-space where the model is inaccurate (Da Silva et al., 2023; Shyam et al., 2019).

We argue that this could greatly benefit MORL algorithms, especially scenarios where the utility function is unknown, and multiple policies have to be learned, as we only need to learn a single model, that can be shared across all policies. This approach has recently shown promising results in the unknown linear utility function scenario, greatly reducing the number of required interactions compared to the model-free variant (Alegre et al., 2023).

More concretely, we envision the combination of model-based approaches with PCN. As explained in Chapter 5, PCN learns its policy based on trajectory executions. When the environment is stochastic, the associated return might be due to chance, not to the quality of the policy. Focusing on these trajectories can then lead to negative outcomes, as the agent tries to repeat unlikely transitions, often resulting in worse outcomes than desired. With a model, the transition probabilities can be learned, and we can estimate the quality of the policy in terms of expectations, by executing multiple trajectories in the model. We have recently experimented with this approach, combining PCN with Wasserstein Autoencoded MDPs (Delgrange et al., 2023), an approach that learns a model of the environment with formal bisimilarity guarantees, i.e., putting bounds on the difference of behavior of the policy in the model and the environment. While learning the model, we are able to learn unexpected events, and discard policies that rely on these events. Moreover, we extend the single-objective guarantees presented in the original work to the multi-objective case (Reymond, Delgrange, et al., 2023). However, the bisimilarity guarantees are bounded by the model's loss function. Learning a model based on a changing set of policies, and thus

on changing trajectories, is a complicated task that can lead to instability issues (i.e., a high loss), as the dataset of trajectories changes over time. Thus, future work is required to reliably learn the model, and prevent these instabilities.

Bibliography

Abdolmaleki, A., Huang, S., Hasenclever, L., Neunert, M., Song, F., Zambelli, M., Martins, M., Heess, N., Hadsell, R., & Riedmiller, M. (2020). A distributional view on multi-objective policy optimization. *International Conference on Machine Learning*, 11–22.

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., & Riedmiller, M. (2018). Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*.

Abdolshah, M., Shilton, A., Rana, S., Gupta, S., & Venkatesh, S. (2019). Multi-objective bayesian optimisation with preferences over objectives. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2019/file/a7b7e4b27722574c611fe91476a50238-Paper.pdf

Abels, A., Roijers, D. M., Lenaerts, T., Nowé, A., & Steckelmacher, D. (2019). Dynamic weights in multi-objective deep reinforcement learning. *Proceedings of the 36th International Conference on Machine Learning*, *97*, 11–20.

Abrams, S., Wambua, J., Santermans, E., Willem, L., Kuylen, E., Coletti, P., Libin, P., Faes, C., Petrof, O., Herzog, S. A., et al. (2021). Modelling the early phase of the belgian covid-19 epidemic using a stochastic compartmental model and studying its implied future trajectories. *Epidemics*, *35*, 100449.

Agarwal, M., Aggarwal, V., & Lan, T. (2022). Multi-objective reinforcement learning with non-linear scalarization. *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 9–17.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., & Bellemare, M. (2021). Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, *34*, 29304–29320.

Alegre, L. N., Bazzan, A. L., Roijers, D. M., Nowé, A., & da Silva, B. C. (2023). Sample-efficient multi-objective learning via generalized policy improvement prioritization. *arXiv preprint arXiv:2301.07784*.

Alegre, L. N., Bazzan, A., & Da Silva, B. C. (2022). Optimistic linear support and successor features as a basis for optimal policy transfer. *International Conference on Machine Learning*, 394–413.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. (2021). What matters for on-policy deep actor-critic methods? a large-scale study. *International conference on learning representations*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., & Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, *30*.

Arrow, K. J., Chenery, H. B., Minhas, B. S., & Solow, R. M. (1961). Capital-labor substitution and economic efficiency. *The review of Economics and Statistics*, 225–250.

Astudillo, R., & Frazier, P. (2020). Multi-attribute bayesian optimization with interactive preference learning. *International Conference on Artificial Intelligence and Statistics*, 4496–4507.

Audibert, J.-Y., Bubeck, S., & Munos, R. (2010). Best arm identification in multi-armed bandits. *COLT*, 41–53.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, *47*, 235–256.

Bailey, N. T. (1975). The mathematical theory of infectious diseases and its applications. In *The mathematical theory of infectious diseases and its applications* (pp. 413–413). Charles Griffin & Company Ltd 5a Crendon Street, High Wycombe, Bucks HP13 6LE.

Bargiacchi, E., Verstraeten, T., Roijers, D., Nowé, A., & Hasselt, H. (2018). Learning to coordinate with coordination graphs in repeated single-stage multi-agent decision problems. *International conference on machine learning*, 482–490.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., & Silver, D. (2017). Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, *30*.

Barreto, A., Hou, S., Borsa, D., Silver, D., & Precup, D. (2020). Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, *117*(48), 30079–30087.

Barrett, L., & Narayanan, S. (2008). Learning all optimal policies with multiple criteria. *Proceedings of the 25th international conference on Machine learning*, 41–47.

Bastani, H., Drakopoulos, K., Gupta, V., Vlachogiannis, I., Hadjicristodoulou, C., Lagiou, P., Magiorkinis, G., Paraskevis, D., & Tsiodras, S. (2021). Efficient and targeted covid-19 border testing via rl. *Nature*, *599*(7883), 108–113.

Belakaria, S., Deshwal, A., Jayakodi, N. K., & Doppa, J. R. (2020). Uncertainty-aware search framework for multi-objective bayesian optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(06), 10044–10052.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*.

Bhatnagar, S., & Lakshmanan, K. (2012). An online actor–critic algorithm with function approximation for constrained markov decision processes. *Journal of Optimization Theory and Applications*, *153*, 688–708.

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol. 4). Springer.

Britton, T., & Lindenstrand, D. (2009). Epidemic modelling: Aspects where stochasticity matters. *Mathematical biosciences*, *222*(2), 109–116.

Brochu, E., Cora, V. M., & De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Burda, Y., Edwards, H., Storkey, A., & Klimov, O. (2019). Exploration by random network distillation. *International Conference on Learning Representations*.

Carbaugh, R. J. (2013). *Contemporary economics: An applications approach*. ME Sharpe.

Castelletti, A., Pianosi, F., & Restelli, M. (2013). A multiobjective reinforcement learning approach to water resources systems operation: Pareto frontier approximation in a single run. *Water Resources Research*, *49*(6), 3476–3486.

Castelletti, A., Pianosi, F., & Restelli, M. (2012). Tree-based fitted q-iteration for multi-objective markov decision problems. *The 2012 International Joint Conference on Neural Networks (IJCNN)*, 1–8.

Chapelle, O., & Li, L. (2011). An empirical evaluation of thompson sampling. *Advances in neural information processing systems*, *24*.

Chaslot, G. M. J., Winands, M. H., Herik, H. J. v. d., Uiterwijk, J. W., & Bouzy, B. (2008). Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation*, *4*(03), 343–357.

Chen, X., Ghadirzadeh, A., Björkman, M., & Jensfelt, P. (2019). Meta-learning for multi-objective reinforcement learning. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 977–983.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf

Chu, W., & Ghahramani, Z. (2005). Preference learning with gaussian processes. *Proceedings of the 22nd international conference on Machine learning*, 137–144.

Couëtoux, A., Hoock, J.-B., Sokolovska, N., Teytaud, O., & Bonnard, N. (2011). Continuous upper confidence trees. *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, 433–445.

Coulom, R. (2007a). Computing "elo ratings" of move patterns in the game of go. *ICGA journal*, *30*(4), 198–208.

Coulom, R. (2007b). Efficient selectivity and backup operators in monte-carlo tree search. *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, 72–83.

Da Silva, F. L., Yang, J., Landajuela, M., Goncalves, A., Ladd, A., Faissol, D., & Petersen, B. (2023). Toward multi-fidelity reinforcement learning for symbolic optimization.

Dabney, W., Ostrovski, G., Silver, D., & Munos, R. (2018). Implicit quantile networks for distributional reinforcement learning. *International conference on machine learning*, 1096–1105.

Das, I., & Dennis, J. E. (1997). A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural optimization*, *14*(1), 63–69.

Daulton, S., Balandat, M., & Bakshy, E. (2020). Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. *Advances in Neural Information Processing Systems*, *33*, 9851–9864.

Deb, K., Agrawal, S., Pratap, A., & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. *International conference on parallel problem solving from nature*, 849–858.

Delgrange, F., Nowe, A., & Perez, G. (2023). Wasserstein auto-encoded MDPs: Formal verification of efficiently distilled RL policies with many-sided guarantees. *The Eleventh*

International Conference on Learning Representations. https://openreview.net/forum?id=JLLTtEdh1ZY

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2017). Density estimation using real NVP. *International Conference on Learning Representations*. https://openreview.net/forum?id=HkpbnH9lx

Doucet, A., Godsill, S., & Andrieu, C. (2000). On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, *10*, 197–208.

Douglas, P. H. (1976). The cobb-douglas production function once again: Its history, its testing, and some new empirical values. *Journal of political economy*, *84*(5), 903–915.

Drugan, M. M., & Nowe, A. (2013). Designing multi-objective multi-armed bandits algorithms: A study. *The 2013 international joint conference on neural networks (IJCNN)*, 1–8.

Eric, B., Freitas, N., & Ghosh, A. (2007). Active preference learning with discrete choice data. *Advances in neural information processing systems*, *20*.

Forgas, J. P. (1995). Mood and judgment: The affect infusion model (aim). *Psychological bulletin*, *117*(1), 39.

Fu, J., Kumar, A., Soh, M., & Levine, S. (2019). Diagnosing bottlenecks in deep q-learning algorithms. *International Conference on Machine Learning*, 2021–2030.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International conference on machine learning*, 1587–1596.

Gábor, Z., Kalmár, Z., & Szepesvári, C. (1998). Multi-criteria reinforcement learning. *ICML*, *98*, 197–205.

Garrido-Merchán, E. C., & Hernández-Lobato, D. (2019). Predictive entropy search for multi-objective bayesian optimization with constraints. *Neurocomputing*, *361*, 50–68.

Gelly, S., & Silver, D. (2011). Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, *175*(11), 1856–1875.

Gordon, N. J., Salmond, D. J., & Smith, A. F. (1993). Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE proceedings F (radar and signal processing)*, *140*(2), 107–113.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International conference on machine learning*, 1861–1870.

Hafner, D., Lillicrap, T., Norouzi, M., & Ba, J. (2020). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

Hayes, C. F., Howley, E., & Mannion, P. (2020). Dynamic thresholded lexicograpic ordering. *Adaptive and Learning Agents Workshop (AAMAS 2020)*.

Hayes, C. F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L. M., Dazeley, R., Heintz, F., Howley, E., Irissappane, A. A., Mannion, P., Nowé, A., Ramos, G., Restelli, M., Vamplew, P., & Roijers, D. M. (2021). A practical guide to multi-objective rl and planning.

Hayes, C. F., Reymond, M., Roijers, D. M., Howley, E., & Mannion, P. (2021). Distributional monte carlo tree search for risk-aware and multi-objective reinforcement learning. *Proceedings of the 20th international conference on autonomous agents and multiagent systems*, 1530–1532.

Hayes, C. F., Reymond, M., Roijers, D. M., Howley, E., & Mannion, P. (2023). Monte carlo tree search algorithms for risk-aware and multi-objective reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, *37*(2), 26.

Hayes, C. F., Verstraeten, T., Roijers, D. M., Howley, E., & Mannion, P. (2021). Expected scalarised returns dominance: A new solution concept for multi-objective decision making. *arXiv preprint arXiv:2106.01048*.

Hayes, C. F., Verstraeten, T., Roijers, D. M., Howley, E., & Mannion, P. (2022). Multi-objective coordination graphs for the expected scalarised returns with generative flow models. *arXiv preprint arXiv:2207.00368*.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI conference on artificial intelligence*, *32*(1).

Hiraoka, K., Yoshida, M., & Mishima, T. (2009). Parallel reinforcement learning for weighted multi-criteria model with adaptive margin. *Cognitive neurodynamics*, *3*(1), 17–24.

Hol, J. D., Schon, T. B., & Gustafsson, F. (2006). On resampling algorithms for particle filters. *2006 IEEE nonlinear statistical signal processing workshop*, 79–82.

Huang, S., Abdolmaleki, A., Vezzani, G., Brakel, P., Mankowitz, D. J., Neunert, M., Bohez, S., Tassa, Y., Heess, N., Riedmiller, M., et al. (2022). A constrained multi-objective reinforcement learning framework. *Conference on Robot Learning*, 883–893.

Ibarz, B., Leike, J., Pohlen, T., Irving, G., Legg, S., & Amodei, D. (2018). Reward learning from human preferences and demonstrations in atari. *Advances in neural information processing systems*, *31*.

Jalalimanesh, A., Haghighi, H. S., Ahmadi, A., Hejazian, H., & Soltani, M. (2017). Multi-objective optimization of radiotherapy: Distributed q-learning and agent-based simulation. *Journal of Experimental & Theoretical artificial intelligence*, *29*(5), 1071–1086.

Kagrecha, A., Nair, J., & Jagannathan, K. (2023). Constrained regret minimization for multi-criterion multi-armed bandits. *Machine Learning*, 1–28.

Kompella, V., Capobianco, R., Jong, S., Browne, J., Fox, S., Meyers, L., Wurman, P., & Stone, P. (2020). Reinforcement learning for optimization of covid-19 mitigation policies. *arXiv preprint arXiv:2010.10560*.

Kumar, A., Peng, X. B., & Levine, S. (2019). Reward-conditioned policies. *arXiv preprint arXiv:1912.13465*.

Kwak, G. H., Ling, L., & Hui, P. (2021). Deep reinforcement learning approaches for global public health strategies for covid-19 pandemic. *PLOS ONE*, *16*(5), 1–15.

Laber, E. B., Lizotte, D. J., & Ferguson, B. (2014). Set-valued dynamic treatment regimes for competing outcomes. *Biometrics*, *70*(1), 53–61.

Laumanns, M., & Ocenasek, J. (2002). Bayesian optimization algorithms for multi-objective optimization. *Parallel Problem Solving from Nature—PPSN VII: 7th International Conference Granada, Spain, September 7–11, 2002 Proceedings 7*, 298–307.

Levine, S., & Koltun, V. (2013). Guided policy search. *International conference on machine learning*, 1–9.

Li, C., & Czarnecki, K. (2018). Urban driving with multi-objective deep reinforcement learning. *arXiv preprint arXiv:1811.08586*.

Li, T., Bolic, M., & Djuric, P. M. (2015). Resampling methods for particle filtering: Classification, implementation, and strategies. *IEEE Signal processing magazine*, *32*(3), 70–86.

Libin, P., Verstraeten, T., Roijers, D. M., Wang, W., Theys, K., & Nowé, A. (2019). Thompson sampling for m-top exploration. *BNAIC/BENELEARN*.

Libin, P. J. K., Moonens, A., Verstraeten, T., Perez-Sanjines, F., Hens, N., Lemey, P., & Nowé, A. (2021). Deep reinforcement learning for large-scale epidemic control. In Y. Dong, G. Ifrim, D. Mladenić, C. Saunders, & S. Van Hoecke (Eds.), *Ecml* (pp. 155–170). Springer International Publishing.

Libin, P. J., Verstraeten, T., Roijers, D. M., Grujic, J., Theys, K., Lemey, P., & Nowé, A. (2018). Bayesian best-arm identification for selecting influenza mitigation strategies. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 456–471.

Libin, P. J., Willem, L., Verstraeten, T., Torneri, A., Vanderlocht, J., & Hens, N. (2021). Assessing the feasibility and effectiveness of household-pooled universal testing to control covid-19 epidemics. *PLoS computational biology*, *17*(3), e1008688.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lim, J., Ryu, S., Kim, J. W., & Kim, W. Y. (2018). Molecular generative model based on conditional variational autoencoder for de novo molecular design. *Journal of cheminformatics*, *10*(1), 1–9.

Lin, Z., Yang, D., Zhao, L., Qin, T., Yang, G., & Liu, T.-Y. (2020). Rd2: Reward decomposition with representation decomposition. *Advances in Neural Information Processing Systems*, *33*, 11298–11308.

Lizotte, D. J., Bowling, M. H., & Murphy, S. A. (2010). Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. *ICML*, *10*, 695–702.

Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., & Wießner, E. (2018). Microscopic traffic simulation using sumo. *The 21st IEEE International Conference on Intelligent Transportation Systems*. https://elib.dlr.de/124092/

Lyle, C., Bellemare, M. G., & Castro, P. S. (2019). A comparative analysis of expected and distributional reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 4504–4511.

Mannion, P., Devlin, S., Duggan, J., & Howley, E. (2018). Reward shaping for knowledge-based multi-objective multi-agent reinforcement learning. *The Knowledge Engineering Review*, *33*, e23.

Mannion, P., Mason, K., Devlin, S., Duggan, J., & Howley, E. (2016). Multi-objective dynamic dispatch optimisation using multi-agent reinforcement learning. *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 1345–1346.

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Moerland, T. M., Broekens, J., Plaat, A., Jonker, C. M., et al. (2023). Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, *16*(1), 1–118.

Moffaert, K. V., & Nowé, A. (2014). Multi-objective reinforcement learning using sets of pareto dominating policies. *JMLR*, *15*(107), 3663–3692.

Morimura, T., Sugiyama, M., Kashima, H., Hachiya, H., & Tanaka, T. (2010). Nonparametric return distribution approximation for reinforcement learning. *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 799–806.

Mossalam, H., Assael, Y. M., Roijers, D. M., & Whiteson, S. (2016). Multi-objective deep reinforcement learning. *CoRR*, *abs/1610.02707*. http://arxiv.org/abs/1610.02707

Murphy, K. P. (2007). Conjugate bayesian analysis of the gaussian distribution. *def*, *1*($2\sigma 2$), 16.

Nasiriany, S., Pong, V., Lin, S., & Levine, S. (2019). Planning with goal-conditioned policies. *Advances in Neural Information Processing Systems*, *32*.

Neil, D., Segler, M., Guasch, L., Ahmed, M., Plumbley, D., Sellwood, M., & Brown, N. (2018). Exploring deep recurrent models with reinforcement learning for molecule design. *6th International Conference on Learning Representations (ICLR), Workshop Track*.

Ohi, A. Q., Mridha, M., Monowar, M. M., Hamid, M., et al. (2020). Exploring optimal control of epidemic spread using rl. *Scientific reports*, *10*(1), 1–19.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., & Peters, J. (2018). An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*.

Painter, M., Lacerda, B., & Hawes, N. (2020). Convex hull monte-carlo tree-search. *Proceedings of the international conference on automated planning and scheduling*, *30*, 217–225.

Paria, B., Kandasamy, K., & Póczos, B. (2020). A flexible framework for multi-objective bayesian optimization using random scalarizations. *Uncertainty in Artificial Intelligence*, 766–776.

Parisi, S., Pirotta, M., & Peters, J. (2017). Manifold-based multi-objective policy search with sample reuse. *Neurocomputing*, *263*, 3–14.

Parisi, S., Pirotta, M., & Restelli, M. (2016). Multi-objective reinforcement learning through continuous pareto manifold approximation. *Journal of Artificial Intelligence Research*, *57*, 187–227.

Parisi, S., Pirotta, M., Smacchia, N., Bascetta, L., & Restelli, M. (2014). Policy gradient approaches for multi-objective sequential decision making. *2014 International Joint Conference on Neural Networks (IJCNN)*, 2323–2330.

Paternain, S., Chamon, L., Calvo-Fullana, M., & Ribeiro, A. (2019). Constrained reinforcement learning has zero duality gap. *Advances in Neural Information Processing Systems*, *32*.

Peng, X. B., Abbeel, P., Levine, S., & Van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)*, *37*(4), 1–14.

Perez, D., Mostaghim, S., Samothrakis, S., & Lucas, S. M. (2014). Multiobjective monte carlo tree search for real-time games. *IEEE Transactions on Computational Intelligence and AI in Games*, *7*(4), 347–360.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., & Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Pianosi, F., Castelletti, A., & Restelli, M. (2013). Tree-based fitted q-iteration for multi-objective markov decision processes in water resource management. *Journal of Hydroinformatics*, *15*(2), 258–270.

Probert, W. J., Lakkur, S., Fonnesbeck, C. J., Shea, K., Runge, M. C., Tildesley, M. J., & Ferrari, M. J. (2019). Context matters: Using reinforcement learning to develop human-readable, state-dependent outbreak response policies. *Philosophical Transactions of the Royal Society B*, *374*(1776), 20180277.

Rădulescu, R., Mannion, P., Roijers, D. M., & Nowé, A. (2020). Multi-objective multi-agent decision making: A utility-based analysis and survey. *Autonomous Agents and Multi-Agent Systems*, *34*(1), 10.

Reddy, M. J., & Kumar, D. N. (2006). Optimal reservoir operation using multi-objective evolutionary algorithm. *Water Resources Management*, *20*, 861–878.

Reymond, M., Delgrange, F., Nowé, A., & Pérez, G. (2023). Wae-pcn: Wasserstein-autoencoded pareto conditioned networks. *Proceedings of the adaptive and learning agents workshop (ALA-23) at AAMAS*. https://alaworkshop2023.github.io/papers/ALA2023_paper_42.pdf

Reymond, M., Eugenio, B., & Nowè, A. (2022). Pareto conditioned networks. *Proceedings of the 21st International Conference on AAMAS (2022)*.

Reymond, M., Hayes, C. F., Steckelmacher, D., Roijers, D. M., & Nowé, A. (2023). Actor-critic multi-objective reinforcement learning for non-linear utility functions. *Autonomous Agents and Multi-Agent Systems*, *37*(2), 23. https://doi.org/10.1007/s10458-023-09604-x

Reymond, M., Hayes, C. F., Willem, L., Rădulescu, R., Abrams, S., Roijers, D. M., Howley, E., Mannion, P., Hens, N., Nowé, A., et al. (2022). Exploring the pareto front of multi-objective covid-19 mitigation policies using reinforcement learning. *arXiv preprint arXiv:2204.05027*.

Reymond, M., & Nowé, A. (2019). Pareto-dqn: Approximating the pareto front in complex multi-objective decision problems. *Proceedings of the adaptive and learning agents workshop (ALA-19) at AAMAS*.

Rhuggenaath, J., Akcay, A., Zhang, Y., & Kaymak, U. (2019). Optimizing reserve prices for publishers in online ad auctions. *2019 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr)*, 1–8.

Riihimäki, J., & Vehtari, A. (2010). Gaussian processes with monotonicity information. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 645–652.

Roijers, D. M., Röpke, W., Nowé, A., & Rădulescu, R. (2021). On following pareto-optimal policies in multi-objective planning and reinforcement learning. *Proceedings of the Multi-Objective Decision Making (MODeM) Workshop*.

Roijers, D. M., Steckelmacher, D., & Nowé, A. (2018). Multi-objective reinforcement learning for the expected utility of the return. *Proceedings of the Adaptive and Learning Agents workshop at FAIM*.

Roijers, D. M., Vamplew, P., Whiteson, S., & Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *JAIR*, *48*, 67–113.

Roijers, D. M., Zintgraf, L. M., Libin, P., Reymond, M., Bargiacchi, E., & Nowé, A. (2021). Interactive multi-objective reinforcement learning in multi-armed bandits with gaussian process utility models. *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*, 463–478.

Roijers, D. M., Zintgraf, L. M., & Nowé, A. (2017). Interactive thompson sampling for multi-objective multi-armed bandits. *Algorithmic Decision Theory: 5th International Conference, ADT 2017, Luxembourg, Luxembourg, October 25–27, 2017, Proceedings 5*, 18–34.

Roijers, D. M., Whiteson, S., & Oliehoek, F. A. (2015). Computing convex coverage sets for faster multi-objective coordination. *Journal of Artificial Intelligence Research*, *52*, 399–443.

Röpke, W., Hayes, C. F., Mannion, P., Howley, E., Nowé, A., & Roijers, D. M. (2023). Distributional multi-objective decision making. *arXiv preprint arXiv:2305.05560*.

Ruiz-Montiel, M., Mandow, L., & Pérez-de-la-Cruz, J.-L. (2017). A temporal difference method for multi-objective reinforcement learning. *Neurocomputing*, *263*, 15–25.

Russo, D. (2016). Simple bayesian algorithms for best arm identification. *Conference on Learning Theory*, 1417–1418.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., et al. (2022). Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, *35*, 36479–36494.

Saisubramanian, S., Kamar, E., & Zilberstein, S. (2021). A multi-objective approach to mitigate negative side effects. *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, 354–361.

Schaul, T., Horgan, D., Gregor, K., & Silver, D. (2015). Universal value function approximators. *International conference on machine learning*, 1312–1320.

Schmidhuber, J. (2019). Reinforcement learning upside down: Don't predict rewards–just map them to actions. *arXiv preprint arXiv:1912.02875*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shen, C., Wang, Z., Villar, S., & Van Der Schaar, M. (2020). Learning for dose allocation in adaptive clinical trials with safety constraints. *International Conference on Machine Learning*, 8730–8740.

Shen, Y., Tobia, M. J., Sommer, T., & Obermayer, K. (2014). Risk-sensitive reinforcement learning. *Neural computation*, *26*(7), 1298–1328.

Shyam, P., Jaśkowski, W., & Gomez, F. (2019). Model-based active exploration. *International conference on machine learning*, 5779–5788.

Sidney, S. (1957). Nonparametric statistics for the behavioral sciences. *The Journal of Nervous and Mental Disease*, *125*(3), 497.

Silver, D., & Veness, J. (2010). Monte-carlo planning in large pomdps. *Advances in neural information processing systems*, *23*.

Sirakaya, E., Petrick, J., & Choi, H.-S. (2004). The role of mood on tourism product evaluations. *Annals of Tourism Research*, *31*(3), 517–539.

Skalse, J., Hammond, L., Griffin, C., & Abate, A. (2022, July). Lexicographic multi-objective reinforcement learning [Main Track]. In L. D. Raedt (Ed.), *Proceedings of the thirty-first international joint conference on artificial intelligence, IJCAI-22* (pp. 3430–3436). International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2022/476

Sui, Y., Gotovos, A., Burdick, J., & Krause, A. (2015). Safe exploration for optimization with gaussian processes. *International conference on machine learning*, 997–1005.

Sui, Y., Zhuang, V., Burdick, J., & Yue, Y. (2018). Stagewise safe bayesian optimization with gaussian processes. *International conference on machine learning*, 4781–4789.

Sun, W., Bagnell, J. A., & Boots, B. (2018). Truncated horizon policy search: Combining reinforcement learning & imitation learning. *arXiv preprint arXiv:1805.11240*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Szandała, T. (2021). Review and comparison of commonly used activation functions for deep neural networks. *Bio-inspired neurocomputing*, 203–224.

Tamar, A., & Mannor, S. (2013). Variance adjusted actor critic algorithms. *arXiv preprint arXiv:1310.3697*.

Tesauro, G. (1988). Connectionist learning of expert preferences by comparison training. *Advances in neural information processing systems*, *1*.

Tesauro, G., Das, R., Chan, H., Kephart, J., Levine, D., Rawson, F., & Lefurgy, C. (2008). Managing power consumption and performance of computing systems using reinforcement learning. *Advances in Neural Information Processing Systems*, 1497–1504.

Tessler, C., Mankowitz, D. J., & Mannor, S. (2019). Reward constrained policy optimization. *International Conference on Learning Representations*. https://openreview.net/forum?id=SkfrvsA9FX

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, *25*(3-4), 285–294.

Torneri, A., Libin, P., Vanderlocht, J., Vandamme, A.-M., Neyts, J., & Hens, N. (2020). A prospect on the use of antiviral drugs to control local outbreaks of covid-19. *BMC medicine*, *18*(1), 1–9.

Torneri, A., Willem, L., Colizza, V., Kremer, C., Meuris, C., Darcis, G., Hens, N., & Libin, P. J. (2021). Controlling sars-cov-2 in schools using repetitive testing strategies (preprint).

Turner, A., Ratzlaff, N., & Tadepalli, P. (2020). Avoiding side effects in complex environments. *Advances in Neural Information Processing Systems*, *33*, 21406–21415.

Turner, A. M., Hadfield-Menell, D., & Tadepalli, P. (2020). Conservative agency via attainable utility preservation. *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 385–391.

Vamplew, P., Dazeley, R., Barker, E., & Kelarev, A. (2009). Constructing stochastic mixture policies for episodic multiobjective reinforcement learning tasks. *AI 2009: Advances in Artificial Intelligence: 22nd Australasian Joint Conference, Melbourne, Australia, December 1-4, 2009. Proceedings 22*, 340–349.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., & Dekker, E. (2011). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning*, *84*(1-2), 51–80.

Vamplew, P., Dazeley, R., & Foale, C. (2017). Softmax exploration strategies for multiobjective reinforcement learning. *Neurocomputing*, *263*, 74–86.

Vamplew, P., Smith, B. J., Källström, J., Ramos, G., Rădulescu, R., Roijers, D. M., Hayes, C. F., Heintz, F., Mannion, P., Libin, P. J., et al. (2022). Scalar reward is not enough: A response to silver, singh, precup and sutton (2021). *Autonomous Agents and Multi-Agent Systems*, *36*(2), 41.

Vamplew, P., Yearwood, J., Dazeley, R., & Berry, A. (2008). On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts. *Australasian joint conference on artificial intelligence*, 372–378.

Van den Burg, G. J. J. (2020). An exploration of thompson sampling. *gertjanvandenburg.com/blog*. https://gertjanvandenburg.com/blog/thompson_sampling/

Van Moffaert, K., Drugan, M. M., & Nowé, A. (2013a). Hypervolume-based multi-objective reinforcement learning. *Evolutionary Multi-Criterion Optimization: 7th International Conference, EMO 2013, Sheffield, UK, March 19-22, 2013. Proceedings 7*, 352–366.

Van Moffaert, K., Drugan, M. M., & Nowé, A. (2013b). Scalarized multi-objective reinforcement learning: Novel design techniques. *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 191–199.

Van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., & Tsang, J. (2017). Hybrid reward architecture for reinforcement learning. *Advances in Neural Information Processing Systems*, *30*.

Verstraeten, T., Bargiacchi, E., Libin, P. J., Helsen, J., Roijers, D. M., & Nowé, A. (2020). Multi-agent thompson sampling for bandit applications with sparse neighbourhood structures. *Scientific reports*, *10*(1), 6728.

Verstraeten, T., Nowé, A., Keller, J., Guo, Y., Sheng, S., & Helsen, J. (2019). Fleetwide data-enabled reliability improvement of wind turbines. *Renewable and Sustainable Energy Reviews*, *109*, 428–437.

Vieillard, N., Pietquin, O., & Geist, M. (2020). Munchausen reinforcement learning. *Advances in Neural Information Processing Systems*, *33*, 4235–4246.

Wallinga, J., Teunis, P., & Kretzschmar, M. (2006). Using data on social contacts to estimate age-specific transmission parameters for respiratory-spread infectious agents. *American journal of epidemiology*, *164*(10), 936–944.

Wan, R., Zhang, X., & Song, R. (2021). Multi-objective model-based reinforcement learning for infectious disease control. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 1634–1644.

Wang, S., Reymond, M., Irissappane, A. A., & Roijers, D. M. (2022). Near on-policy experience sampling in multi-objective reinforcement learning. *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, 1756–1758.

Wang, W., & Sebag, M. (2013). Hypervolume indicator and dominance reward based multi-objective monte-carlo tree search. *Machine learning*, *92*, 403–429.

White, D. (1982). Multi-objective infinite-horizon discounted markov decision processes. *Journal of mathematical analysis and applications*, *89*(2), 639–647.

Wiering, M. A., & De Jong, E. D. (2007). Computing optimal stationary policies for multi-objective markov decision processes. *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 158–165.

Wiering, M. A., Withagen, M., & Drugan, M. M. (2014). Model-based multi-objective reinforcement learning. *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 1–6.

Willem, L., Abrams, S., Libin, P. J., Coletti, P., Kuylen, E., Petrof, O., Møgelmose, S., Wambua, J., Herzog, S. A., Faes, C., et al. (2021). The impact of contact tracing and household bubbles on deconfinement strategies for covid-19. *Nature communications*, *12*(1), 1–9.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3-4), 229–256.

Wirth, C., Akrour, R., Neumann, G., Fürnkranz, J., et al. (2017). A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, *18*(136), 1–46.

Wirth, C., Fürnkranz, J., & Neumann, G. (2016). Model-free preference-based reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, *30*(1).

Wray, K., Zilberstein, S., & Mouaddib, A.-I. (2015). Multi-objective mdps with conditional lexicographic reward preferences. *Proceedings of the AAAI Conference on Artificial Intelligence*, *29*(1).

Wray, K. H., & Zilberstein, S. (2015). Multi-objective pomdps with lexicographic reward preferences. *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Xu, J., Tian, Y., Ma, P., Rus, D., Sueda, S., & Matusik, W. (2020). Prediction-guided multi-objective reinforcement learning for continuous robot control. *International conference on machine learning*, 10607–10616.

Yahyaa, S. Q., Drugan, M. M., & Manderick, B. (2014). Annealing-pareto multi-objective multi-armed bandit algorithm. *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 1–8.

Yahyaa, S. Q., & Manderick, B. (2015). Thompson sampling for multi-objective multi-armed bandits problem. *ESANN*.

Yamaguchi, T., Nagahama, S., Ichikawa, Y., & Takadama, K. (2019). Model-based multi-objective reinforcement learning with unknown weights. *Human Interface and the Management of Information. Information in Intelligent Systems: Thematic Area, HIMI 2019, Held as Part of the 21st HCI International Conference, HCII 2019, Orlando, FL, USA, July 26-31, 2019, Proceedings, Part II 21*, 311–321.

Yang, D., Zhao, L., Lin, Z., Qin, T., Bian, J., & Liu, T.-Y. (2019). Fully parameterized quantile function for distributional reinforcement learning. *Advances in neural information processing systems*, *32*.

Yang, R., Sun, X., & Narasimhan, K. (2019). A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In *Advances in neural information processing systems 32* (pp. 14610–14621). Curran Associates, Inc.

Yliniemi, L., & Tumer, K. (2016). Multi-objective multiagent credit assignment in reinforcement learning and nsga-ii. *Soft Computing*, *20*(10), 3869–3887.

Zhang, P., Chen, X., Zhao, L., Xiong, W., Qin, T., & Liu, T.-Y. (2021). Distributional reinforcement learning for multi-dimensional reward functions. *Advances in Neural Information Processing Systems*, *34*, 1519–1529.

Zhang, R., & Golovin, D. (2020). Random hypervolume scalarizations for provable multi-objective black box optimization. *International Conference on Machine Learning*, 11096–11105.

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P., & Irving, G. (2019). Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.

Zintgraf, L. M., Kanters, T. V., Roijers, D. M., Oliehoek, F., & Beau, P. (2015). Quality assessment of morl algorithms: A utility-based approach. *Benelearn 2015: proceedings of the 24th annual ML conference of Belgium and the Netherlands*.

Zintgraf, L. M., Roijers, D. M., Linders, S., Jonker, C. M., & Nowé, A. (2018). Ordered preference elicitation strategies for supporting multi-objective decision making. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 1477–1485.

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, *3*(4), 257–271.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & Da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*, *7*(2), 117–132.

Zoghi, M., Whiteson, S., Munos, R., & Rijke, M. (2014). Relative upper confidence bound for the k-armed dueling bandit problem. *International conference on machine learning*, 10–18.

Zur, R. M., Jiang, Y., Pesce, L. L., & Drukker, K. (2009). Noise injection for training artificial neural networks: A comparison with weight decay and early stopping. *Medical physics*, *36*(10), 4810–4818.

Publications

Hayes, C., Radulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L., Dazeley, R., Heintz, F., Howley, E., Irissappane, A., Mannion, P., Nowe, A., De Oliveira Ramos, G., Restelli, M., Vamplew, P., & Roijers, D. (2023). A brief guide to multi-objective reinforcement learning and planning: Jaamas track [The 22nd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023 ; Conference date: 29-05-2023 Through 02-06-2023]. *The 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023)*. https://aamas2023.soton.ac.uk

Reymond, M., Delgrange, F., Nowe, A., & Pérez, G. (2023, May). Wae-pcn: Wasserstein-autoencoded pareto conditioned networks [2023 Adaptive and Learning Agents Workshop at AAMAS, ALA 2023 ; Conference date: 29-05-2023 Through 30-05-2023]. In F. Cruz, C. Hayes, C. Wang, & C. Yates (Eds.), *Proc. of the adaptive and learning agents workshop (ala 2023)* (15th ed., pp. 1–7, Vol. https://alaworkshop2023.github.io/). https://alaworkshop2023.github.io

Reymond, M., Hayes, C., Steckelmacher, D., Roijers, D., & Nowe, A. (2023). Actor-critic multi-objective reinforcement learning for non-linear utility functions [Funding Information: Conor F. Hayes is funded by the University of Galway Hardiman Scholarship. This research was supported by funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program. Publisher Copyright: © 2023, Springer Science+Business Media, LLC, part of Springer Nature. Copyright: Copyright 2023 Elsevier B.V., All rights reserved.]. *Autonomous Agents and Multi-Agent Systems*, *37*(2). https://doi.org/https://doi.org/10.1007/s10458-023-09604-x

Avalos, R., Reymond, M., Nowe, A., & Roijers, D. (2022a). Local advantage networks for cooperative multi-agent reinforcement learning [Funding Information: Raphaël Avalos was supported by the FWO (grant 11F5721N). This research was supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" program. Publisher Copyright: © 2022 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved. Copyright: Copyright 2022 Elsevier B.V., All rights reserved.; 21st International Conference on Autonomous Agents and Multi-agent System, AAMAS ; Conference date: 09-05-2022 Through 13-05-2022]. *International Con-*

ference on Autonomous Agents and Multiagent Systems, AAMAS 2022, 1524–1526. https://aamas2022-conference.auckland.ac.nz

Avalos, R., Reymond, M., Nowe, A., & Roijers, D. (2022b). Local advantage networks for multi-agent reinforcement learning in dec-pomdps. *Proc. of the Adaptive and Learning Agents Workshop (ALA 2023), https://ewrl.wordpress.com/past-ewrl/ewrl15-2022/*, 1–17.

Hayes, C., Radulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L., Dazeley, R., Heintz, F., Howley, E., Irissappane, A., Mannion, P., Nowe, A., De Oliveira Ramos, G., Restelli, M., Vamplew, P., & Roijers, D. (2022). A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, *36*(1). https://doi.org/10.1007/s10458-022-09552-y

Reymond, M., Bargiacchi, E., & Nowe, A. (2022). Pareto conditioned networks [21st International Conference on Autonomous Agents and Multi-agent System, AAMAS ; Conference date: 09-05-2022 Through 13-05-2022]. *The 21st International Conference on Autonomous Agents and Multiagent Systems*, 1110–1118. https://aamas2022-conference.auckland.ac.nz

Reymond, M., Hayes, C., Willem, L., Radulescu, R., Abrams, S., Roijers, D., Howley, E., Mannion, P., Hens, N., Nowe, A., & Libin, P. Exploring the pareto front of multi-objective covid-19 mitigation policies using reinforcement learning [21st European Conference on Computational Biology, ECCB 2022 ; Conference date: 18-09-2022 Through 21-09-2022]. English. In: 21st European Conference on Computational Biology, ECCB 2022 ; Conference date: 18-09-2022 Through 21-09-2022. 2022, September. https://eccb2022.org/

Wang, S., Reymond, M., Irissappane, A., & Roijers, D. (2022). Near on-policy experience sampling in multi-objective reinforcement learning [21st International Conference on Autonomous Agents and Multi-agent System, AAMAS ; Conference date: 09-05-2022 Through 13-05-2022]. *The 21st International Conference on Autonomous Agents and Multiagent Systems*, 1756–1758. https://aamas2022-conference.auckland.ac.nz

Hayes, C., Reymond, M., Roijers, D., Howley, E., & Mannion, P. (2021). Distributional monte carlo tree search for risk-aware and multi-objective reinforcement learning [The 20th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2021 ; Conference date: 03-05-2021 Through 07-05-2021]. *The 20th International Conference on Autonomous Agents and Multiagent Systems*, 1518–1520. https://aamas2021.soton.ac.uk/

Reymond, M., Hayes, C., Roijers, D., Steckelmacher, D., & Nowe, A. Actor-critic multi-objective reinforcement learning for non-linear utility functions. [Multi-Objective Decision Making Workshop 2021, MODeM 2021 ; Conference date: 14-07-2021 Through 16-07-2021]. English. In: Multi-Objective Decision Making Workshop

2021, MODeM 2021 ; Conference date: 14-07-2021 Through 16-07-2021. 2021, July. http://modem2021.cs.nuigalway.ie/

Roijers, D., Zintgraf, L., Libin, P., Reymond, M., Bargiacchi, E., & Nowe, A. (2021). Interactive multi-objective reinforcement learning in multi-armed bandits with gaussian process utility models [ECML PKDD: Joint European Conference on Machine Learning and Knowledge Discovery in Databases
 ; Conference date: 14-09-2020 Through 18-09-2020]. In F. Hutter, K. Kersting, J. Lijffijt, & I. Valera (Eds.), *Ecml-pkdd 2020: Proceedings of the 2020 european conference on machine learning and principles and practice of knowledge discovery in databases*. Springer. https://doi.org/10.1007/978-3-030-67664-3_28

Reymond, M., & Nowe, A. (2019). Pareto-dqn: Approximating the pareto front in complex multi-objective decision problems [null ; Conference date: 13-05-2019 Through 14-05-2019]. *Proceedings of the Adaptive and Learning Agents Workshop 2019 (ALA-19) at AAMAS*. https://ala2019.vub.ac.be

Nevens, J., Radulescu, R., Reymond, M., Van Eecke, P., Efthymiadis, K., & Beuls, K. (2018, November). Hybrid ai for visual question answering on clevr [30th Benelux Conference on Artificial Intelligence, BNAIC 2018 ; Conference date: 08-11-2018 Through 09-11-2018]. In *Bnaic 2018 preproceedings* (pp. 171–172). https://bnaic2018.nl

Reymond, M., Patyn, C., Radulescu, R., Nowe, A., & Deconinck, G. (2018). Reinforcement learning for demand response of domestic household appliances [Adaptive Learning Agents Workshop 2018, ALA-18 ; Conference date: 14-07-2018 Through 15-07-2018]. *Proceedings of the Adaptive Learning Agents Workshop 2018 (ALA-18)*, 18–25. http://ala2018.it.nuigalway.ie/

Appendix A

Appendix for Chapter 3

A.1 Additional results

We show the plots for Deep Sea Treasure, using a known linear function, for weights $w_0 = 0, 0.1, \ldots, 1$, in Figure A.1.

A.2 Hyperparameters

All hyperparameters used for all these experiments are listed in Tables A.1, A.2.

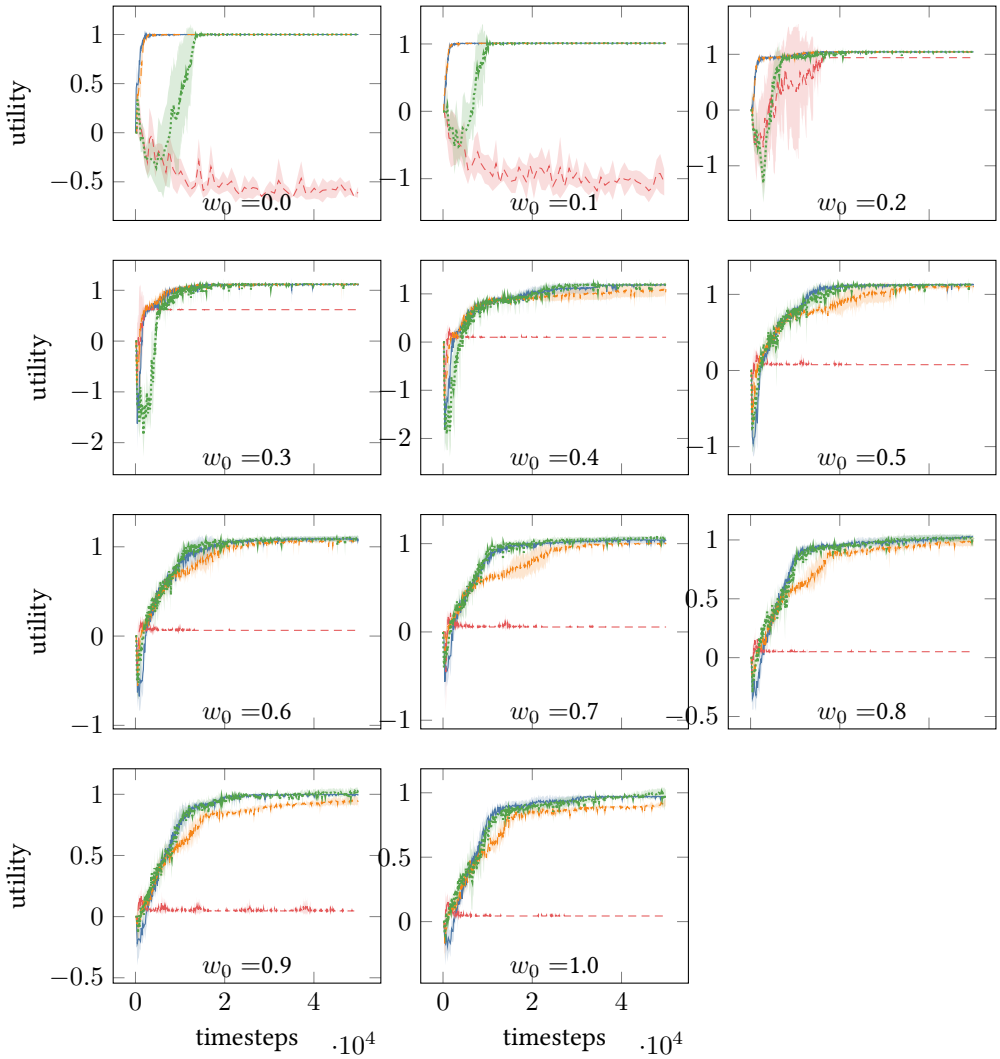


Figure A.1: Deep Sea Treasure with a linear utility function, for weights $w_0 = 0, 0.1, \ldots, 1$.

Table A.1: Hyperparameters for the Split, Deep-sea-treasure and Minecart environments.

	Split	**Deep-Sea-Treasure**	**Minecart**
		Common	
lr	0.001	0.001	0.0003
γ	1.00	0.95	1.00
timesteps	$6,000$	$1,000,000$	$20,000,000$
neurons (actor)	$(26, 20, 2)$	$(132, 50, 4)$	Figure **??**, $(20, 20, 6)$
non-linearity	Tanh	Tanh	Tanh
clip-grad-norm	None	50	50
		MOCAC	
value-coef	0.5	0.5	0.5
entropy-coef	0.01	0.1	0.1
update every	1	10	200
neurons (critic)	$(26, 50, 121)$	$(132, 50, 50, 121)$	Figure **??**, $(20, 20, 121)$
c	11	11	11
V_{MIN}	$(-1, -1)$	$(0, -20)$	$(0, -4)$
V_{MAX}	$(10, 10)$	$(100, 0)$	$(1.5, 0)$
	MOAC	**A2C**	
value-coef	0.5	0.5	0.5
entropy-coef	0.01	0.1	0.1
update every	1	10	200
neurons (critic)	$(26, 50, 1)$	$(132, 50, 50, 1)$	$(26, 20, 20, 1)$

Table A.2: Hyperparameters for the experiments on MiniRandom and Fishwood. Since the input-size of the neural networks depend on the conditioning, the *neurons* rows are variable.

	MiniRandom	**Fishwood**
	Common	
lr	0.001	0.001
γ	1.00	1.00
timesteps	$10,000$	$1,000,000$
neurons (actor)	$(5 + 2, 50, 3)$	$(2 + 2, 50, 2)$
non-linearity	Tanh	Tanh
clip-grad-norm	50	50
	MOCAC	
value-coef	0.5	0.5
entropy-coef	0.01	0.01
update every	1	5
neurons (critic)	$(5 + 2, 50, 50, 121)$	$(2 + 2, 50, 50, 121)$
c	7	11
V_{MIN}	$(0, 0)$	$(0, 0)$
V_{MAX}	$(7, 7)$	$(4, 7)$

Appendix B

Appendix for Chapter 6

B.1 Stochastic Compartmental Model

In this work we utilize the compartmental model proposed by (Abrams et al., 2021) and extend the model to a multi-objective Markov decision process (MOMDP). Figure 6.1 shows a visual depiction of the compartment model. The flows of the deterministic model are defined by a set of ordinary differential equations, outlined in Section 6.2.1.

Intervening in the spread of the virus by, for example, reducing social contacts or government interventions introduces uncertainty in the further course of the outbreak. Therefore, to understand how this uncertainty affect the spread of the disease we introduce a stochastic component model which can model the uncertainty generated by interventions in social contacts.

By formulating the set of differential equations defined in Section 6.2.1, as a chain-binomial, we can obtain stochastic trajectories from this model (Bailey, 1975). A chain-binomial model assumes a stochastic model where infected individuals are generated by some underlying probability distribution. For the stochastic model we consider a time interval $(t, t+h]$, where h is defined as the length between two consecutive time points. Similar to (Abrams et al., 2021), in this work we set $h = \frac{1}{24}$. (Abrams et al., 2021) define the set of differential equations as a chain binomial as follows:

$$
\begin{aligned}
\mathbf{S}_{t+h}(k) &= \mathbf{S}_t(k) - \mathbf{E}_{new,t+h}(k),\\
\mathbf{E}_{t+h}(k) &= \mathbf{E}_t(k) + \mathbf{E}_{new,t+h}(k) - \mathbf{I}^{presym}_{new,t+h}(k),\\
\mathbf{I}^{presym}_{t+h}(k) &= \mathbf{I}^{presym}_t(k) + \mathbf{I}^{presym}_{new,t+h}(k) - \mathbf{I}^{asym}_{new,t+h}(k) - \mathbf{I}^{mild}_{new,t+h}(k),\\
\mathbf{I}^{asym}_{t+h}(k) &= \mathbf{I}^{asym}_t(k) + \mathbf{I}^{asym}_{new,t+h}(k) - \mathbf{R}^{asym}_{new,t+h}(k),\\
\mathbf{I}^{mild}_{t+h}(k) &= \mathbf{I}^{mild}_t(k) + \mathbf{I}^{mild}_{new,t+h}(k) - \mathbf{I}^{sev}_{new,t+h}(k) - \mathbf{R}^{mild}_{new,t+h}(k),\\
\mathbf{I}^{sev}_{t+h}(k) &= \mathbf{I}^{sev}_t(k) + \mathbf{I}^{sev}_{new,t+h}(k) - \mathbf{I}^{hosp}_{new,t+h}(k) - \mathbf{I}^{icu}_{new,t+h}(k),\\
\mathbf{I}^{hosp}_{t+h}(k) &= \mathbf{I}^{hosp}_t(k) + \mathbf{I}^{hosp}_{new,t+h}(k) - \mathbf{D}^{hosp}_{new,t+h}(k) - \mathbf{R}^{hosp}_{new,t+h}(k),\\
\mathbf{I}^{icu}_{t+h}(k) &= \mathbf{I}^{icu}_t(k) + \mathbf{I}^{icu}_{new,t+h}(k) - \mathbf{D}^{icu}_{new,t+h}(k) - \mathbf{R}^{icu}_{new,t+h}(k),\\
\mathbf{D}_{t+h}(k) &= \mathbf{D}_t(k) + \mathbf{D}^{hosp}_{new,t+h}(k) + \mathbf{D}^{icu}_{new,t+h}(k),\\
\mathbf{R}_{t+h}(k) &= \mathbf{R}_t(k) + \mathbf{R}^{asym}_{new,t+h}(k) + \mathbf{R}^{mild}_{new,t+h}(k) + \mathbf{R}^{hosp}_{new,t+h}(k) + \mathbf{R}^{icu}_{new,t+h}(k)
\end{aligned}
$$

where,

$$
\begin{aligned}
\mathbf{E}_{new,t+h} &\sim \textit{Binomial}\left(\mathbf{S}_t(k), p^*_t(k) = 1 - \{1 - p^*_t(k)\}^{\mathbf{I}_t}\right),\\
p^*_t(k) &= 1 - exp\left[-h\sum_{k'=1}^{K}\beta_{asym}(k,k')\{\mathbf{I}^{asym}_t(k')\} + \beta_{sym}(k,k')\{\mathbf{I}^{mild}_t(k') + \mathbf{I}^{sev}_t(k')\}\right],\\
\mathbf{I}^{presym}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{presym}_t(k), 1 - exp(-hp(k)\theta)\right),\\
\mathbf{I}^{mild}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{presym}_t(k), 1 - exp\left[-h\{1 - p(k)\}\theta\right]\right),\\
\mathbf{I}^{sev}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{mild}_t(k), 1 - exp\{-h\psi(k)\}\right),\\
\mathbf{I}^{hosp}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{sev}_t(k), 1 - exp\{-h\phi_1(k)\omega(k)\}\right),\\
\mathbf{I}^{icu}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{sev}_t(k), 1 - exp\left[-h\{1 - \phi_1(k)\}\omega(k)\right]\right),\\
\mathbf{D}^{hosp}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{hosp}_t(k), 1 - exp\{-h\tau_1(k)\}\right),\\
\mathbf{D}^{icu}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{icu}_t(k), 1 - exp\{-h\tau_2(k)\}\right),\\
\mathbf{R}^{asym}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{asym}_t(k), 1 - exp\left(-h\delta_2(k)\right)\right),\\
\mathbf{R}^{hosp}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{hosp}_t(k), 1 - exp\{-h\delta_3(k)\}\right),\\
\mathbf{R}^{icu}_{new,t+h}(k) &\sim Binomial\left(\mathbf{I}^{icu}_t(k), 1 - exp\{-h\delta_4(k)\}\right).
\end{aligned}
$$

Given MOBelCov also calculates new hospitalizations, $\mathbf{H}^{new}$, we define $\mathbf{H}^{new}$ for the stochastic compartmental model as follows:

$$
\mathbf{H}^{new}_{t+h}(k) = \mathbf{H}^{new}_t(k) + \mathbf{I}^{hosp}_{new,t+h}(k).
$$

For more details on this model and the chain-binomial representation of the differential equations, we refer the reader to the work of (Abrams et al., 2021).

To create a version of MOBelCov with a stochastic transition function, M, we utilize the stochastic compartmental model outlined above. Given the transitions within the compartmental model are derived by an underlying probability distribution it is possible to utilize the stochastic compartmental model transitions for MOBelCov. As previously outlined in Section 6.2.1 the contact matrix $\hat{C}$ applied the model state s_m progresses the model and returns a new model state s'_m. Given the underlying model dynamics are governed in a probabilistic manner, the model returns s'_m stochastically. Therefore, it is possible to use this process as a stochastic transition function, M, for MOBelCov.

B.1.1 Modelling interventions

In order to model different types of interventions, we follow (Abrams et al., 2021). Firstly, to consider distinct exit scenarios, we alter the social contact matrices to reflect a contact reduction in a particular age group. Secondly, we assume that compliance to the interventions is gradual and model this using a logistic compliance function. We use the logistic compliance function in function of time t,

$$c(t, t_I) = \frac{\exp(\beta_0^* + \beta_1^*(t - t_I))}{1 + \exp(\beta_0^* + \beta_1^*(t - t_I))}, \tag{B.1}$$

where t_I indicates the time the intervention started. We initialize β_1^* to the value estimated in by Abrams et al. and choose $\beta_0^* = -5$, as an intercept to have $c(t) = 0$ for $t = 0$, in correspondence with Figure F2 in the Supplementary Information of (Abrams et al., 2021).

B.2 Additional results

B.2.1 Comparison of coverage sets learned on ODE and Binomial models

The coverage sets displayed in Figure 6.3 correspond to PCN trained on the MOBelCov model, which is stochastic. To show that PCN copes with the stochasticity of this model, we compare these coverage sets with the ones learned on the ODE model, which is deterministic.

Results are shown in Figure B.1. First, in a similar fashion as Figure 6.3, we show the interpolated average coverage set for different budget setting, with PCN trained on the ODE model. We observe similar trends as for the Binomial model. Next, for each budget setting, we compare the interpolated average coverage set of the Binomial setting with the ODE setting. We observe similar coverage set, regardless of the budget setting, indicating that PCN is able to cope with stochastic transitions.

B.2.2 Comparison of coverage sets learned on $\mathbf{R}_{\text{ARH}}$ and $\mathbf{R}_{\text{ARI}}$

In Section 6.4.3, we show that we can learn all trade-offs between social burden and hospitalizations, while using the attack rate over infections as reward function.

Results are shown in Figure B.2. We observe that the learned coverage sets are similar, regardless of the budget setting. Still, the coverage set of when trained on $\mathbf{R}_{\text{ARI}}$ is system-

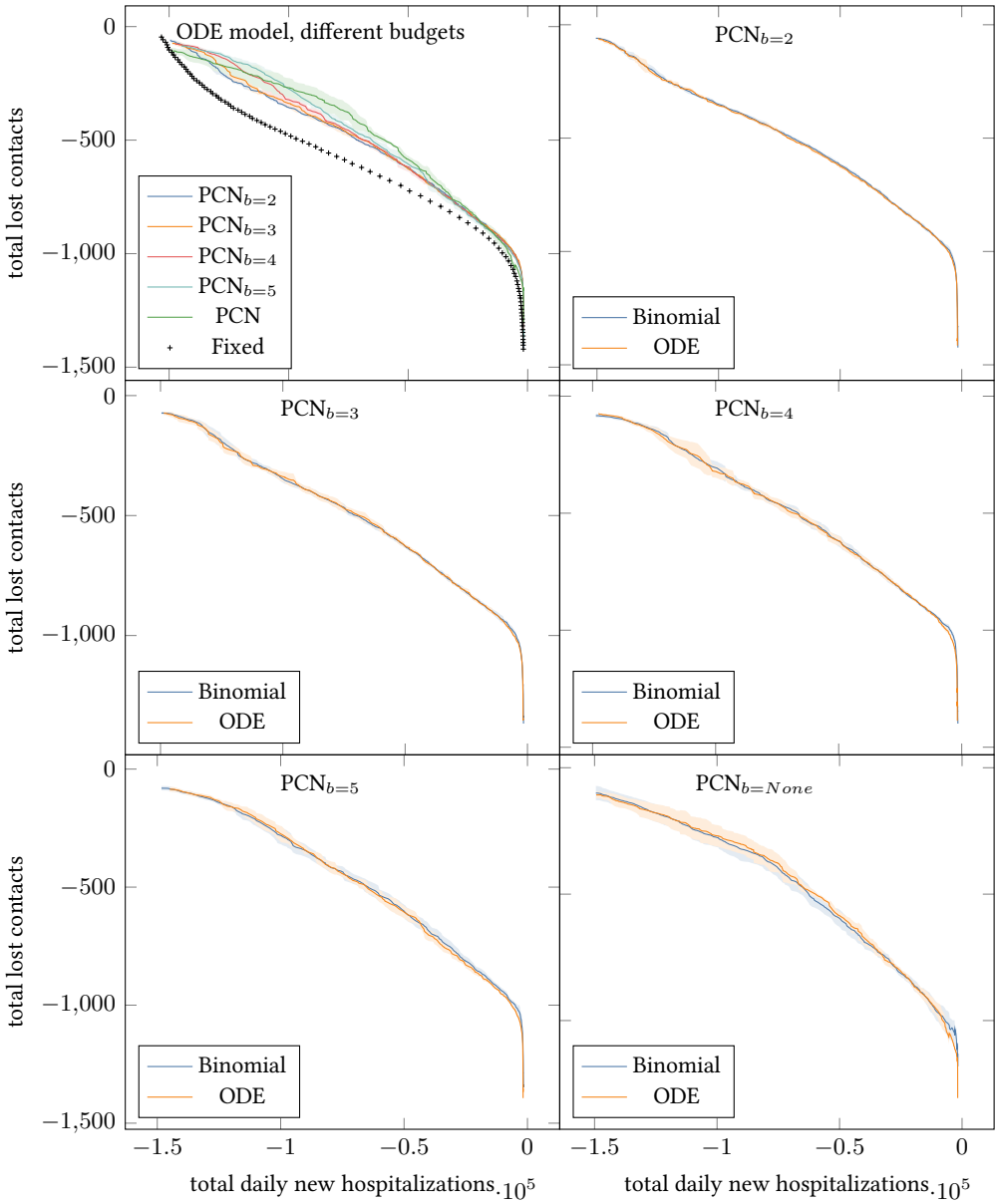


Figure B.1: Comparison of learned coverage sets when PCN interacts with the ODE (i.e., deterministic) variant of the compartment model. We observe that, regardless of the chosen budget, PCN learns a similar coverage set on both variants of the model, indicating that it is able to cope with the stochasticity present in the Binomial variant.

atically dominated by the one when trained on $\mathbf{R}_{\text{ARH}}$. While hospitalizations and infec-

hyper-parameter	value	grid-search
learning rate	0.001	
total training timesteps	300000	
batch size	256	$256, 1024$
model updates	50	
episodes between updates	10	
ER size (in episodes)	1000	$400, 500, 1000$
initial random episodes	200	$50, 200$
exploration noise	0.1	$0, 0.1, 0.2$
desired return noise	0.05	$0, 0.05, 0.1, 0.2$
reward scaling	$[10000, 100]$	

Table B.1: The different hyperparameters used by our extension of PCN. The right-most column also shows, when applicable, the different values tried during grid-search.

tions are highly correlated, they differ in terms of age-groups. Older age-groups are more susceptible to be hospitalized after being infected, but they do not form the majority of the population. For trade-offs where infections and social burden need to be balanced, the proportional reductions target different social environments than for trade-offs balancing hospitalizations and social burden. For example, the work environment is majorly comprised of individuals with a more robust immune system, reducing the social contact in this environment greatly affects the number of infections, but has a lesser impact on the number of hospitalizations.

B.2.3 Experiment parameters

We used the same hyperparameters across all experiments. Each experiment resulted in 10 independent trials. Finally, we performed a grid-search over possible hyperparameter values. All hyperparameters used and their possible values explored during grid-search are displayed in Table B.1.

B.2.4 Neural network architecture

Next to the hyperparameter search, we also performed a grid search over 4 different neural network architectures. All the architectures have the same structure. We use a compartment embedding sc_{emb}, a social contact matrix embedding sm_{emb} and a school-holidays embedding sh_{emb} that take as inputs the compartment, the previous p_w, p_s, p_l values (as they fully define the SCM $\hat{C}$) and a boolean flag for school holidays, respectively. All these embeddings have a same-sized embedding of 64, which are multiplied together to form the full state embedding. This state-embedding is used as input for another network, s_{emb}. Additionally, we use a common embedding c_{emb} for the concatenation of the desired return and horizon. Finally, the results of s_{emb} and c_{emb} are multiplied together, before passing through a fully connected network fc that has 3 outputs, one for p_w, p_s, p_l respectively.

All the architectures of the different components are displayed in Table B.2. The variant used in all experiments is `dense-big`.

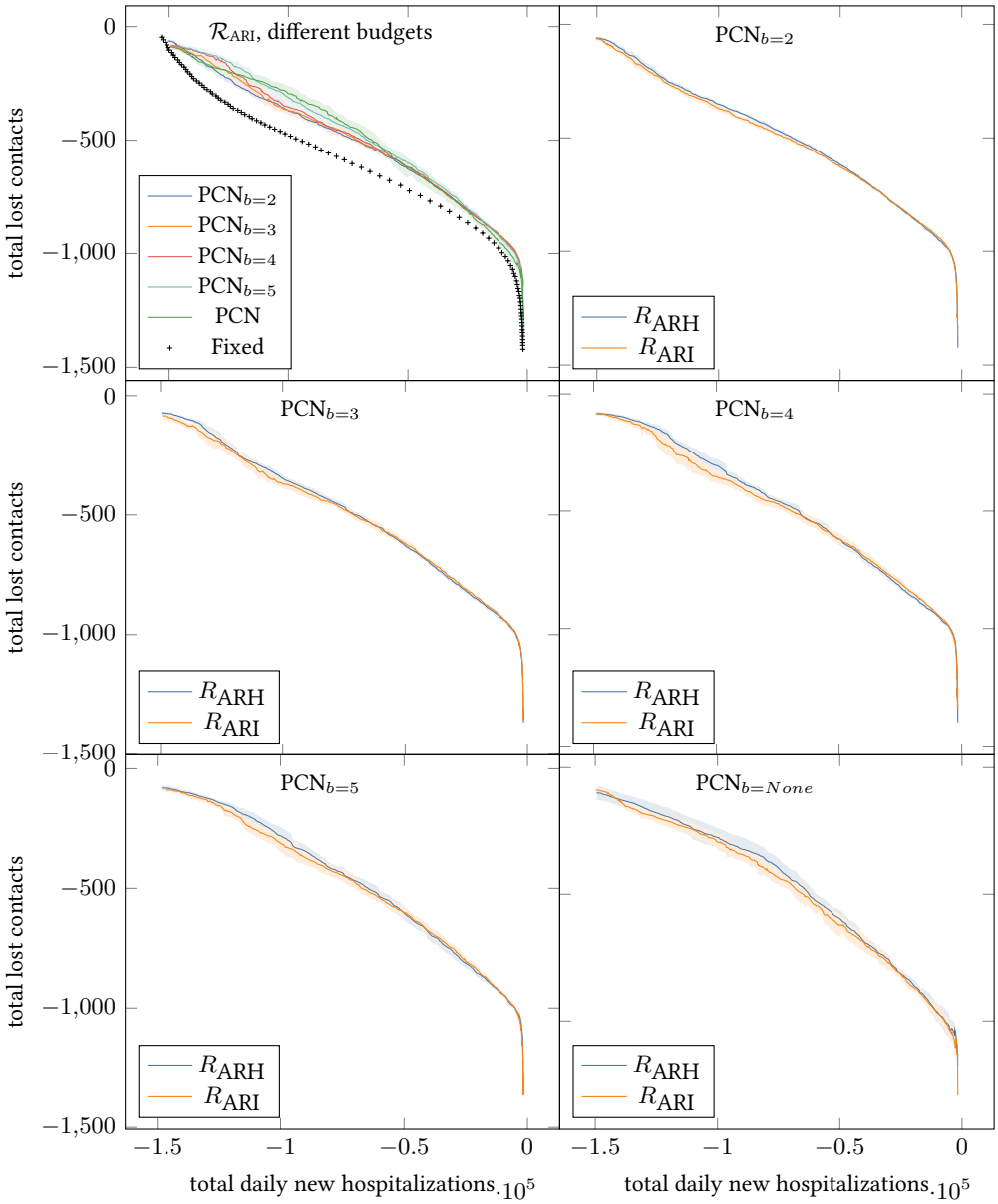


Figure B.2: Comparison of learned coverage sets when PCN learns using the attack rate of infections $\mathbf{R}_{\text{ARI}}$.

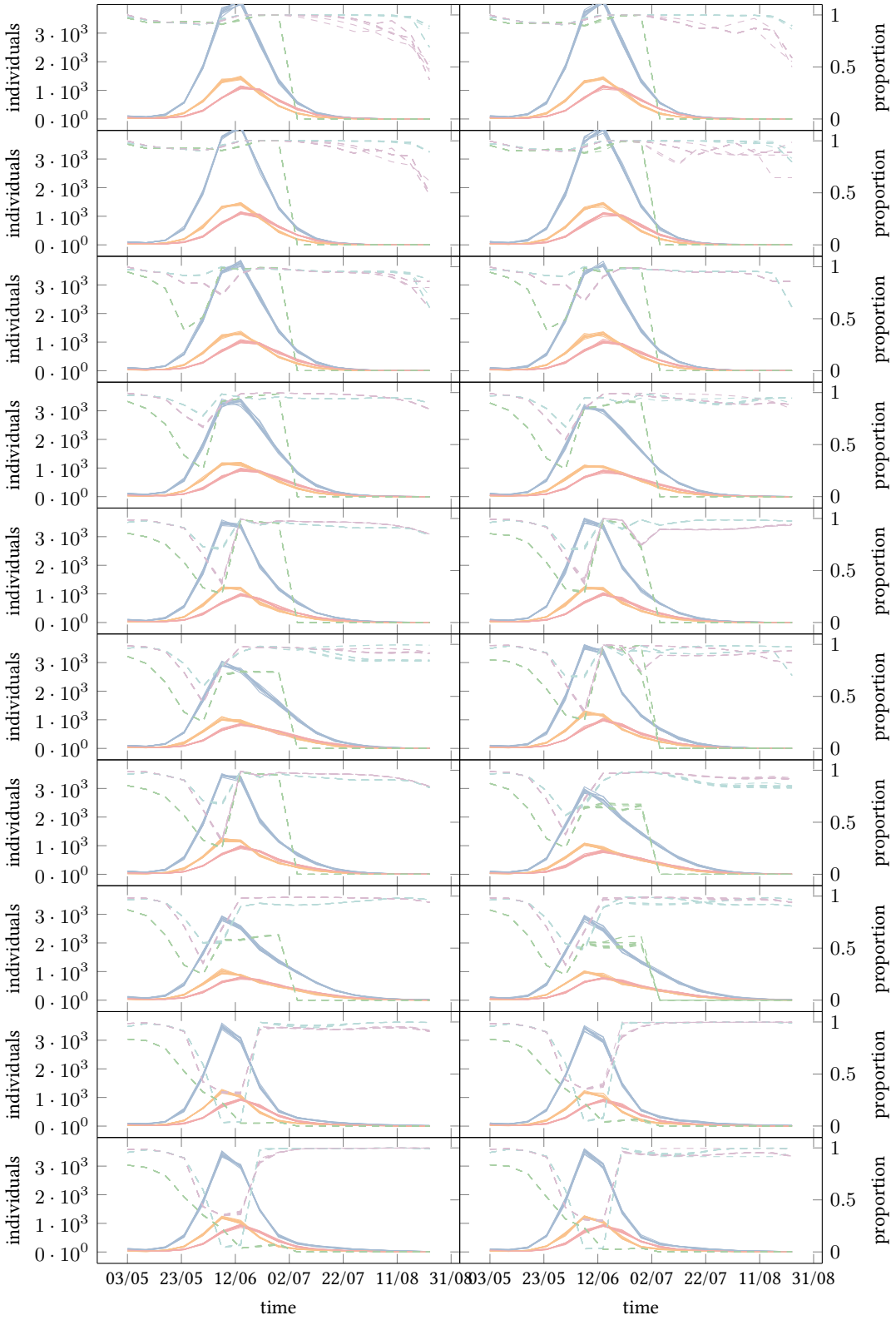
B.2.5 Policy executions

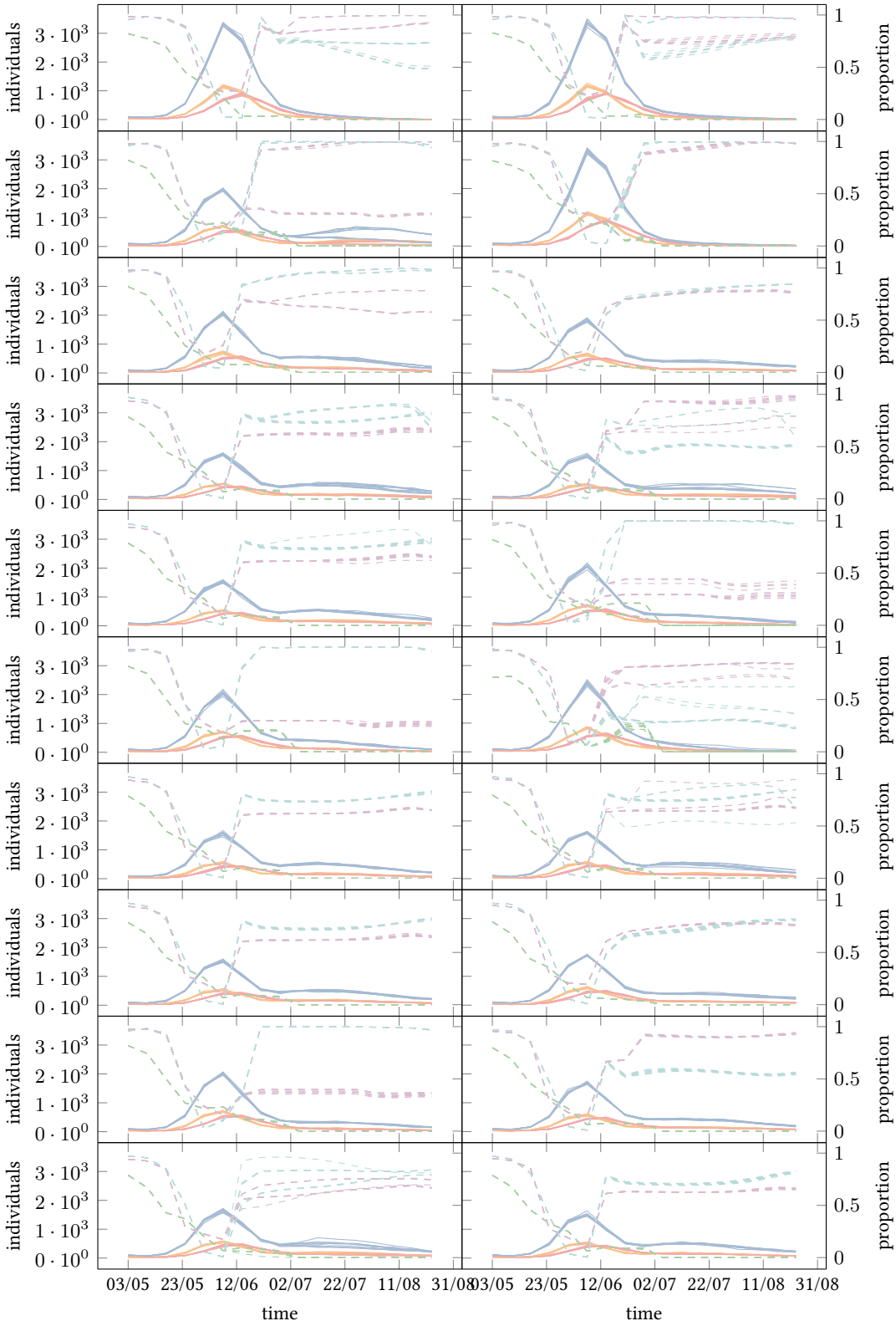
Depending on the budget, PCN learns a coverage set containing more than 150 different policies. To gain a better insight about their behavior, and how they differ from each other,

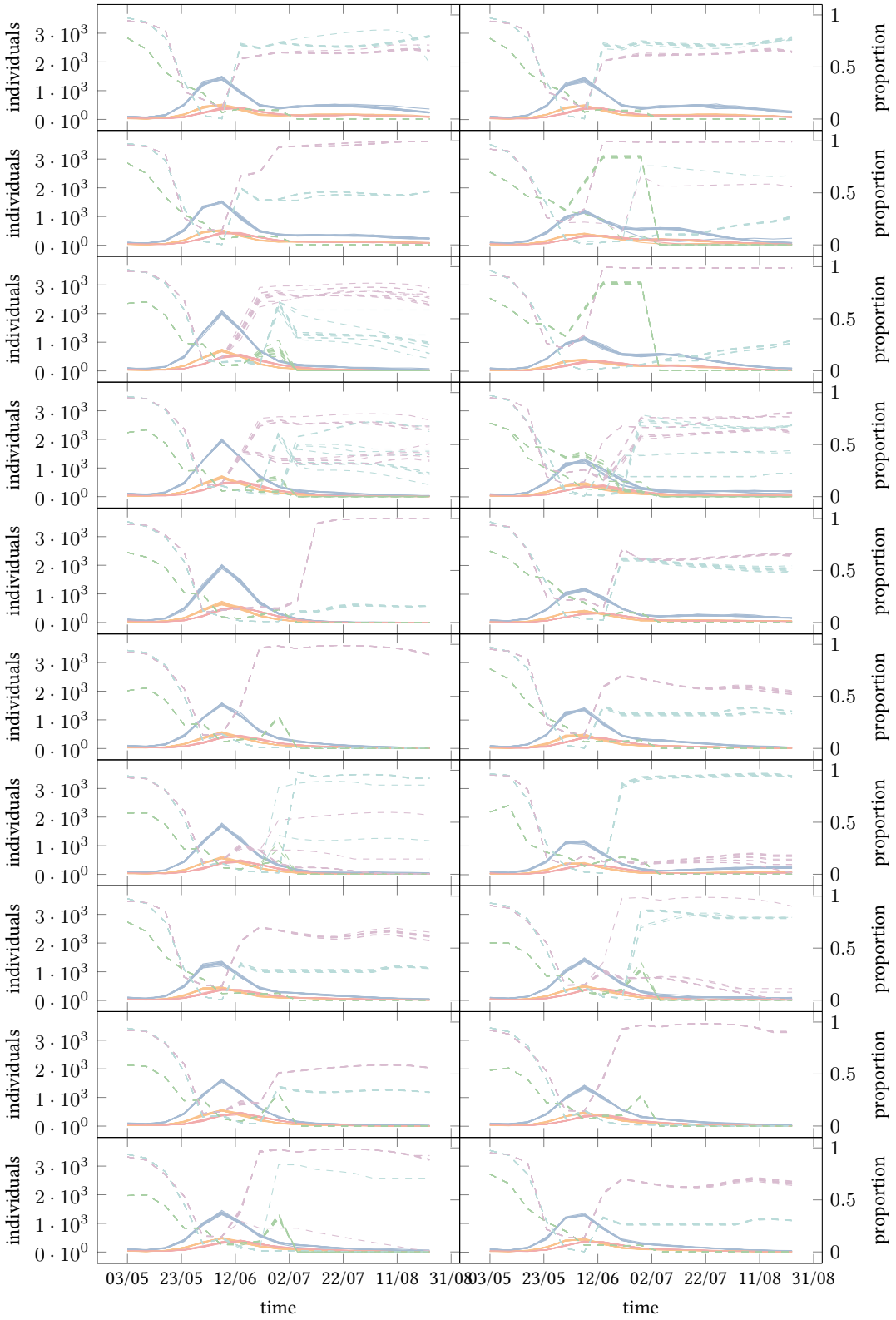
variant	sc_{emb}	sm_{emb}	sh_{emb}	s_{emb}	c_{emb}	fc
conv1d-small	conv1d(10,20) relu conv1d(20,20) relu linear(100,64) sigmoid	linear(3,64) sigmoid	linear(1,64) sigmoid	linear(64,64) sigmoid	linear(3,64) sigmoid	linear(64,64) relu linear(64,3)
conv1d-big	conv1d(10,20) relu conv1d(20,20) relu linear(100,64) sigmoid	linear(3,64) relu linear(64,64) sigmoid	linear(1,64) relu linear(64,64) sigmoid	linear(64,64) relu	linear(3,64) sigmoid	linear(64,64) relu linear(64,3)
dense-small	linear(130,64) sigmoid	linear(3,64) sigmoid	linear(1,64) sigmoid	linear(64,64) sigmoid	linear(3,64) sigmoid	linear(64,64) relu linear(64,3)
dense-big	linear(130,64) relu linear(64,64) sigmoid	linear(3,64) relu linear(64,64) sigmoid	linear(1,64) relu linear(64,64) sigmoid	linear(64,64) relu	linear(3,64) sigmoid	linear(64,64) relu linear(64,3)

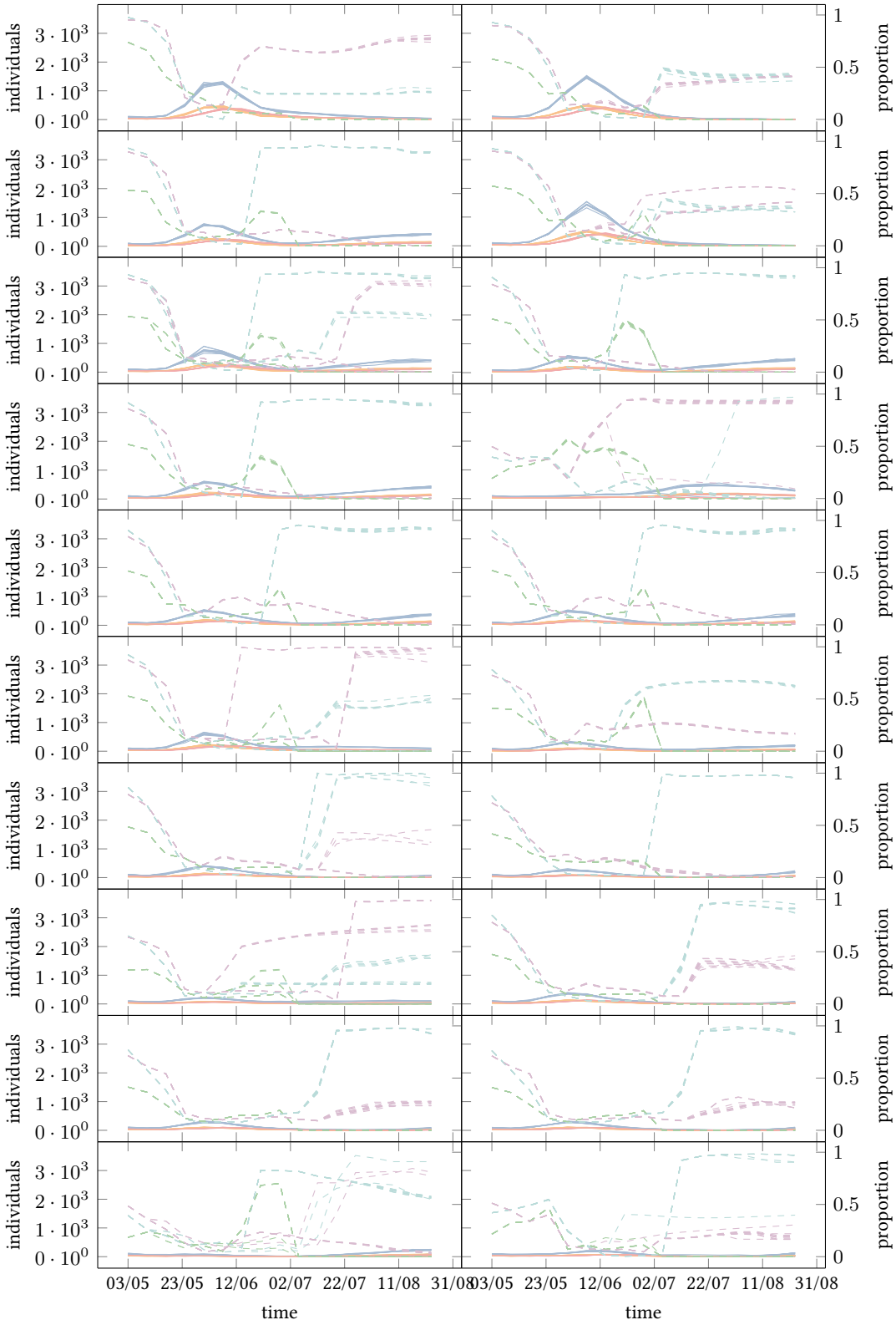
Table B.2: The 4 different neural network architectures explored for our experiments. All the displayed results use the `dense-big` variant.

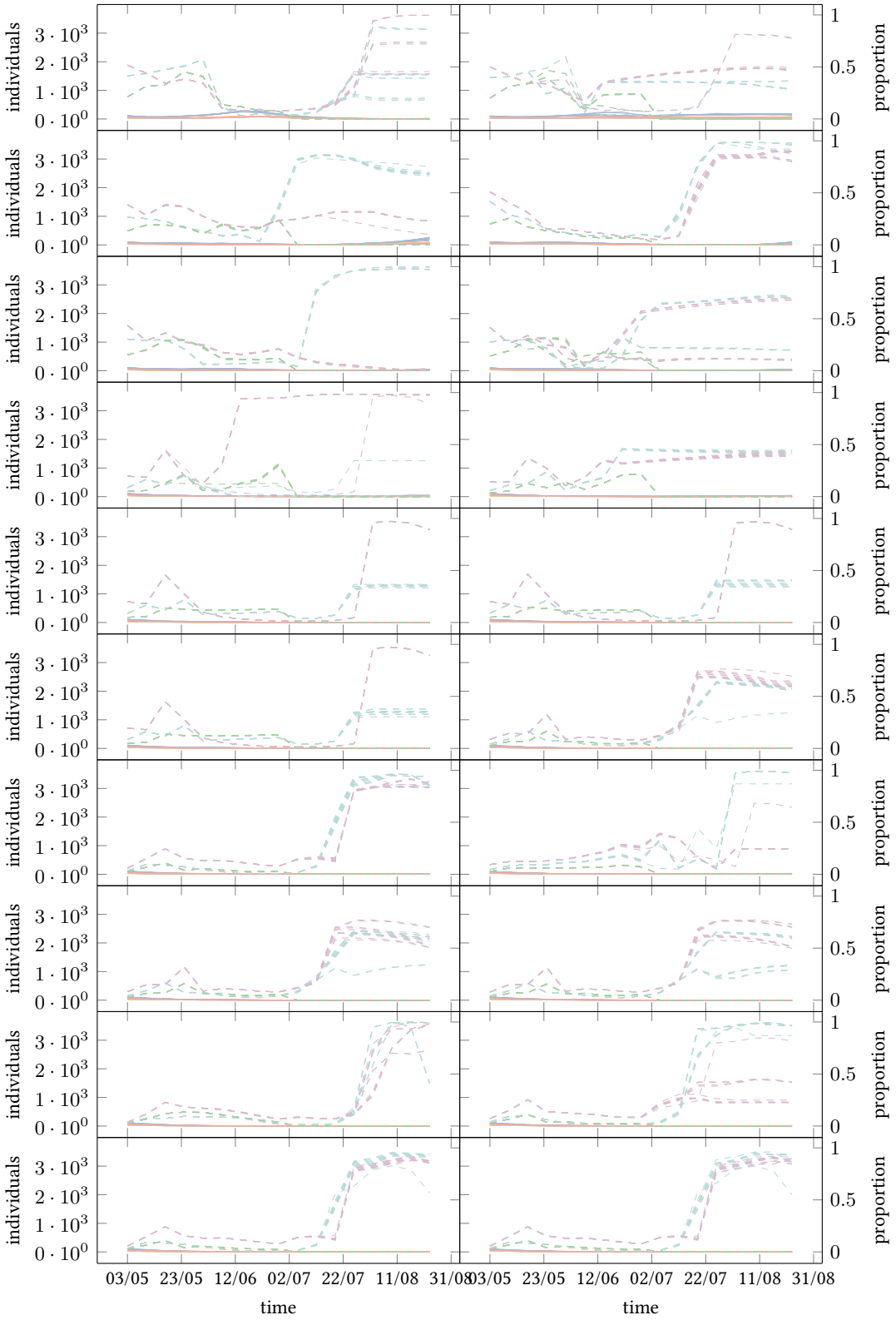
we plot executions of each learned policy for a budget of 5 in Fig. B.3-B.8. The plots are displayed from the least restrictive policy in terms of social burden to the most restrictive one. Since the MOBelCov model is stochastic, we show 10 executions of the same policy, on each plot.


individuals
proportion
$3 \cdot 10^3$
$2 \cdot 10^3$
$1 \cdot 10^3$
$0 \cdot 10^0$
1
0.5
0
03/05
23/05
12/06
02/07
22/07
11/08
31/08
time


individuals
proportion
$3 \cdot 10^3$
$2 \cdot 10^3$
$1 \cdot 10^3$
$0 \cdot 10^0$
1
0.5
0
03/05 23/05 12/06 02/07 22/07 11/08 31/08
time


individuals
proportion
$3 \cdot 10^3$
$2 \cdot 10^3$
$1 \cdot 10^3$
$0 \cdot 10^0$
1
0.5
0
03/05 23/05 12/06 02/07 22/07 11/08 31/08
time


individuals
proportion
$3 \cdot 10^3$
$2 \cdot 10^3$
$1 \cdot 10^3$
$0 \cdot 10^0$
1
0.5
0
03/05 23/05 12/06 02/07 22/07 11/08 31/08
time


individuals
proportion
$3 \cdot 10^3$
$2 \cdot 10^3$
$1 \cdot 10^3$
$0 \cdot 10^0$
1
0.5
0
03/05 23/05 12/06 02/07 22/07 11/08 31/08
time

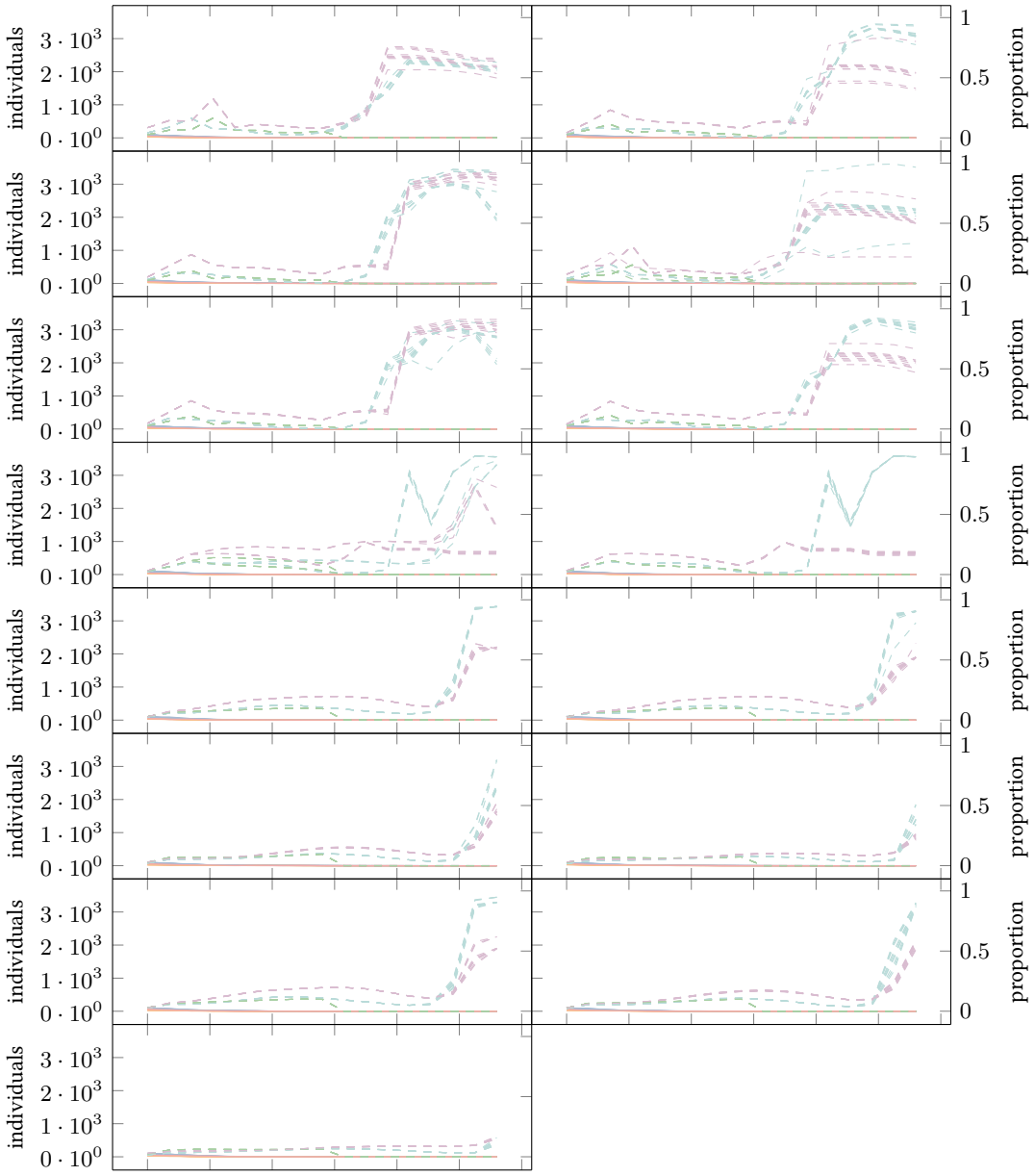


Figure B.8: Execution of policies 100 to 114, with a budget of 5.