LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

MASTER THESIS

Multi-Objective Reinforcement Learning using Evolutionary Algorithms for Diverse Policy Selection

Clarissa Kümhof

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR MOBILE UND VERTEILTE SYSTEME

MASTER THESIS

Multi-Objective Reinforcement Learning using Evolutionary Algorithms for Diverse Policy Selection

Clarissa Kümhof

Aufgabensteller:	Prof. Dr. Claudia Linnhoff-Popien
Betreuer:	Philipp Altmann Maximilian Wolfgang Zorn
Abgabetermin:	07. März 2025

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 07. März 2025

(Unterschrift des Kandidaten)

Abstract

In recent years, Reinforcement Learning (RL) has gained significant attention as a framework for sequential decision-making, where an agent learns to maximize cumulative rewards through interaction with an environment. A subcategory of RL, Multi-Objective Reinforcement Learning (MORL), focuses on optimizing multiple conflicting objectives simultaneously. In such cases, achieving a balance between these objectives is often necessary. However, traditional MORL methods often lack the ability to maintain diversity among solutions and typically produce only a single policy as output. Especially in dynamic environments, different trade-offs between objectives may be required at different times. To tackle these challenges, this thesis introduces *EMOS — an Evolutionary Multi-Objective Selector* — that integrates RL with Evolutionary Algorithms (EAs) to solve multi-objective problems more effectively. By using EAs to evolve a population of policies, the method aims to approximate the Pareto front by offering a population of diverse solutions rather than selecting a single policy. Building on concepts from *Evolutionary Reinforcement Learning (ERL)* and *Prediction-Guided MORL (PGMORL)*, the proposed framework integrates the actor-critic methodology *Proximal Policy Optimization*, by incorporating trained policies into the evolutionary population to combine exploration through EAs with exploitation through policy gradient updates. This hybrid approach allows for flexible decision making in the inference stage, during which policies can be selected dynamically based on different strategies or goals (e.g., minimizing regret and uncertainty) at each time step. EMOS is benchmarked against the state-of-the-art algorithms *Concave-Augmented Pareto Q-Learning (CAPQL)* and *Generalized Policy Improvement (GPI)* to evaluate both its methodological strengths and its empirical performance. An open source repository with MORL baselines is used to ensure reproducibility and compatibility with common MORL environments like *MuJoCo*. EMOS outperforms PGMORL in expected utility and hypervolume, particularly in *mo-halfcheetah-v4*, which demonstrates diverse solution generation, although its sparsity is higher due to its exploratory nature. Additionally, EMOS achieves these results while requiring fewer computational resources, as it does not learn a prediction model and works with fewer policies in the population compared to PGMORL. This makes EMOS a competitive option even with limited resources. EMOS demonstrates a stable approach to policy evolution with a well-distributed Pareto front in certain environments, but its performance still falls behind CAPQL and GPI, which achieve higher hypervolume and expected utility.

Zusammenfassung

In den letzten Jahren hat das Reinforcement Learning (RL) als Rahmen für sequentielle Entscheidungsfindung erheblich an Aufmerksamkeit gewonnen. Dabei lernt ein Agent, durch Interaktion mit einer Umgebung kumulative Belohnungen zu maximieren. Eine Unterkategorie von RL, das Multi-Objective Reinforcement Learning (MORL), konzentriert sich darauf, mehrere widersprüchliche Ziele gleichzeitig zu optimieren. In solchen Fällen ist es oft notwendig, ein Gleichgewicht zwischen diesen Zielen zu finden. Allerdings fehlt es traditionellen MORL-Methoden häufig an der Fähigkeit, eine Vielfalt an Lösungen aufrechtzuerhalten, da sie typischerweise nur eine einzelne Strategie als Ergebnis liefern. Besonders in dynamischen Umgebungen können zu unterschiedlichen Zeitpunkten verschiedene Zielabwägungen erforderlich sein. Um diese Herausforderungen zu bewältigen, stellt diese Arbeit *EMOS — einen Evolutionary Multi-Objective Selector —* vor, der RL mit Evolutionary Algorithms (EAs) kombiniert, um Multi-Objective-Probleme effizienter zu lösen. Durch den Einsatz von EAs zur Evolution einer Population von Strategien zielt die Methode darauf ab, die Pareto Front zu approximieren, indem sie eine Vielfalt an Lösungen bereitstellt, anstatt nur eine einzelne Strategie auszuwählen. Aufbauend auf Konzepten aus dem *Evolutionary Reinforcement Learning (ERL)* und *Prediction-Guided MORL (PGMORL)* integriert das vorgeschlagene Framework die Actor-Critic-Methodologie *Proximal Policy Optimization*, wobei trainierte Strategien in die evolutionäre Population aufgenommen werden, um Exploration durch EAs mit Exploitation durch *Policy-Gradient-Updates* zu kombinieren. Dieser hybride Ansatz ermöglicht eine flexible Entscheidungsfindung in der Inferenzphase, in der Strategien dynamisch basierend auf verschiedenen Zielen (z.B. Minimierung von Reue und Unsicherheit) zu jedem Zeitpunkt ausgewählt werden können. EMOS wird gegen die aktuellen Algorithmen *Concave-Augmented Pareto Q-Learning (CAPQL)* und *Generalized Policy Improvement (GPI)* getestet, um sowohl seine methodischen Stärken als auch seine empirische Leistung zu bewerten. Ein Open-Source-Repository mit MORL-Baselines wird verwendet, um die Reproduzierbarkeit und Kompatibilität mit gängigen MORL-Umgebungen wie *MuJoCo* sicherzustellen. EMOS übertrifft PGMORL hinsichtlich erwartetem Nutzen und Hypervolumen, insbesondere in *mo-halfcheetah-v4*, was die Erzeugung vielfältiger Lösungen demonstriert. Aufgrund seiner explorativen Natur weist es jedoch eine höhere Streuung auf. Darüber hinaus erzielt EMOS diese Ergebnisse mit geringerem Rechenaufwand, da es kein Vorhersagemodell erlernt und mit weniger Strategien in der Population arbeitet als PGMORL. Dadurch ist EMOS auch bei begrenzten Ressourcen eine wettbewerbsfähige Option. EMOS zeigt einen stabilen Ansatz zur Evolution von Strategien mit einer gut verteilten Pareto Front in bestimmten Umgebungen, bleibt jedoch hinter CAPQL und GPI zurück, die höhere Hypervolumen und erwartete Nutzwerte erzielen.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Philipp and Max, for their invaluable support and guidance throughout this research. Their constructive feedback in our weekly meetings helped shape this thesis, and I am incredibly grateful for the opportunity to deepen my knowledge in this field through their expertise.

A huge thank you goes to my partner, Michael, who not only sat with me every week, giving me someone to discuss my progress with, but also stood by me during the more challenging moments. His unwavering belief in me, emotional support, and inspiration meant more than I can express.

I am also thankful to my friends for their encouragement, especially Emily, whose love for proofreading and well-timed distractions helped me navigate the more stressful periods of this journey. Her support went beyond words — even baking me banana bread when I was overwhelmed, which was both a thoughtful gesture and a much-needed comfort.

Contents

1. Introduction

Reinforcement Learning (RL) has become a relevant research area due to its ability to solve complex decision-making problems. In real-world applications, many tasks involve multiple conflicting objectives, making Multi-Objective Reinforcement Learning (MORL) an important extension. Research in MORL continuously explores new approaches to improve learning efficiency, policy diversity, and adaptability. This thesis contributes to these efforts by integrating Evolutionary Algorithms (EAs) and specific policy selection mechanisms into MORL with the goal of addressing key limitations of existing methods. The first section of this work provides an overview of the motivation, contributions, and structure of this thesis.

1.1. Motivation

In recent years, MORL has gained significant attention due to its ability to optimize multiple, often conflicting, objectives simultaneously. MORL problems differ from conventional RL problems in having two or more objectives to be achieved by the agent, each with its own associated reward signal. As described by Vamplew et al. (2011), especially problems where the objectives are in conflict are considered, because otherwise if all objectives are either directly related or completely independent, they can easily be combined into a single objective and a policy can be found that can maximize all of them [VDB+11]. However, in real-world decision-making problems where objectives are in conflict, any policy must either maximize only one objective, or represent a trade-off between the conflicting objectives. As Hayes et al. (2022) and Zhou et al. (2023) highlight, real-world decision-making tasks are inherently complex, requiring trade-offs between multiple, often conflicting objectives: "Most RL algorithms aim to optimize a single objective. However, in real life, multiple conflicting objectives must be balanced" [ZDA23], and "Real-world sequential decision-making tasks are generally complex, requiring trade-offs between multiple, often conflicting, objectives" [HRB+22]. Some examples of real-world applications include (1) autonomous vehicles where the balance between safety (avoiding accidents) and efficiency (optimizing travel time or comfort) is necessary. These objectives are often conflicting, especially in complex traffic scenarios during which taking risks may save time but endanger passengers. For example, when approaching a yellow traffic light at an intersection, an autonomous vehicle must decide whether to brake or accelerate, i.e., balancing safety, travel time, energy efficiency, and passenger comfort. Braking increases safety but may inconvenience passengers and extend travel time, while accelerating reduces delay but raises the risk of running a red light or causing discomfort. MORL enables the vehicle to dynamically weigh these trade-offs based on the situation, traffic conditions, and passenger preferences. (2) In healthcare, automated treatment plans must balance objectives such as maximizing treatment efficiency and minimizing side effects. Highly effective treatments may have severe side effects, therefore a meticulous decision making process is essential. The examples of autonomous

driving and healthcare illustrate the importance of MORL, as these applications require decision-making systems that can dynamically balance competing objectives rather than optimizing for a single goal.

Despite their growing importance, existing MORL approaches face key challenges that limit their effectiveness in dynamic and uncertain environments. Most traditional MORL algorithms struggle with maintaining diversity in solutions, typically producing a single policy as an output. This policy is often selected based on a predefined user preference, a knee point, or similar criteria. However, this rigid selection process can lead to suboptimal flexibility because different trade-offs between objectives may be preferable under different conditions. This limitation is especially problematic in dynamic environments where objectives change over time and require adaptive decision-making systems that can handle uncertainty and competing objectives beyond static, single-policy solutions. Real-world applications require models that can flexibly navigate trade-offs rather than committing to a fixed preference in advance.

To address these limitations, this thesis proposes an *Evolutionary Multi-Objective Selector (EMOS)* framework, which integrates EAs into MORL while updating selected policies according to specific criteria via policy gradients. The key advantage of this approach is its ability to generate and maintain a diverse population of policies and ensure better coverage of the Pareto front while providing a wide range of solutions that can adapt to different conditions. Unlike traditional methods that require a single policy to be chosen at the end of training, this approach allows for a well-trained population of policies to be retained, and therefore offers greater flexibility in decision-making.

The interaction between EAs and policy selection is the key. By applying evolutionary operations such as mutation, crossover, and selection, the population can be guided toward specific areas of interest on the Pareto front. Instead of predefining a preference vector, this approach allows for an adaptive selection of policies, which makes it more robust to changes in the environment. A diverse population can support various policies even during inference, which eliminates the need to train specifically on a user-defined preference vector. This mechanism dynamically chooses the best policy at each time step based on the current objective. This way, the population can be flexibly trained without explicit preferences, with the goal of fully covering the Pareto front. For example, at one moment, policy A might be the best choice, while at another, policy B might be more suitable. This adaptability is particularly beneficial in scenarios where uncertainty and regret are critical factors and consequently enables the system to respond more effectively to changing conditions. The EMOS framework relies on two fundamental components, which are roughly explained in section 1.2.

In summary, integrating EAs and a specific policy selection mechanism into MORL offers a robust and flexible solution to the challenges of traditional MORL methods. By maintaining diversity and enabling dynamic policy selection, this approach significantly improves adaptability and decision-making in multi-objective environments. Rather than relying on a single, preselected policy, this method allows for a more comprehensive exploration of trade-offs, ultimately leading to more effective and context-aware decision-making systems.

1.2. Contributions

This work develops an algorithm called EMOS (Evolutionary Multi-Objective Selector) following the two main contributions: (1) A dynamic policy selection mechanism that not only guides the training process by determining which policy to update but also supports flexible policy selection during inference. This allows the system for adaptability and prioritization by adapting its strategies in real-time and prioritizing objectives such as safety or achieving a balance between competing goals. Heuristics like regret and uncertainty are used here. (2) A training framework that promotes diversity in the population of policies along the Pareto front. By integrating complex evolutionary mechanisms into policy gradient updates, EMOS prevents local optima and improves the performance across different trade-offs.

1.3. Thesis Structure

This thesis is organised as follows. Initially, a comprehensive background on RL, MORL, and EAs is provided in section 2. It establishes the foundational concepts necessary for understanding the research. Section 3 gives a literature review, categorizes existing MORL algorithms, highlights two algorithms on which EMOS is based, and two more algorithms EMOS is compared against. The methods used to setup the EMOS algorithm for this thesis are described in section 4, followed by the implementation details including benchmarks and metrics used in experiments in section 5. Finally, after presenting, discussing, and comparing the results in section 6, the last section concludes the thesis and suggests future research directions.

2. Background

RL provides a framework for training agents to make sequential decisions by optimizing a specific reward. This section introduces key RL concepts, while starting with the fundamental principles of learning through interaction, the exploration-exploitation trade-off, and policy optimization techniques. Extending RL to multi-objective settings, MORL aims to optimize multiple conflicting objectives simultaneously. The concept of Pareto fronts are discussed, which represent the trade-offs between objectives and serve as a fundamental tool for evaluating MORL solutions. Finally, EAs are introduced, which play a key role in enhancing policy diversity and exploration in RL. These background concepts lay the foundation for understanding the methodologies and comparisons explored in this work.

2.1. Reinforcement Learning (RL)

RL as a subset of machine learning is about an agent interacting with an environment through actions, learning an optimal policy, by trial and error, and receiving feedback in the form of rewards, for sequential decision making problems with the goal of improving its behaviour over time [Li17].

First of all, a typical RL process includes an agent that interacts with the environment: Let $\mathcal{A}$ be a set of actions and let $\mathcal{O}$ be a set of observations. An agent A can be given via a policy function $\pi : \mathcal{O} \to \mathcal{A}$. Given a time series of observations $\langle o_t \rangle_{t \in \mathcal{Z}}$ for some time space $\mathcal{Z}$ the agent can thus generate a time series of actions $\langle a_t \rangle_{t \in \mathcal{Z}}$ by applying $a_t = \pi(o_t)$ [GZLP23]. The environment is the system that the agent interacts with and aims to control. It reacts to the agent's actions, changes its state, and provides feedback in the form of rewards. A state s_t is a representation of the environment at a particular time step t.

A standard RL setting is formalized as a Markov Decision Process (MDP) and consists of an agent interacting with an environment over a number of discrete time steps. An MDP is a 5-tuple, $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where $\mathcal{S}$ is the set of all states, $\mathcal{A}$ is the set of all actions, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function with $r_t = R(s_t, a_t, s_{t+1})$, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ is the transition probability function with $P(s_{t+1}|s, a)$ being the probability of transitioning into state s_{t+1} if one start in state s and take action a using its policy π, and $\gamma \in [0; 1] \subset \mathbb{R}$ is called a discount factor. The intuition behind using a discount γ is that there is no certainty about the future rewards.

The central RL problem is expressed by:

$$\pi^*(s_t) = \arg\max_{a \in \mathcal{A}} V^*(s_{t+1}),$$

with π^* being the optimal policy and V being a value function. In RL, the value of a state s, or state-action pair, is a measure of the expected future rewards that can be obtained by following a certain policy. A value function — used, one way or another, in almost every RL algorithm [Ach18] — provides a way to evaluate how good a particular state or action is. They attempt to find a policy that maximized the discounted return by maintaining a set of estimates of expected discounted returns for some policy — usually either the current (on-policy) or the optimal (off-policy) one.

The expected reward of a policy π being executed starting from state s is given via π's value function:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \cdot \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s, \pi(s)) \cdot V^\pi(s_{t+1}).$$

The optimal policy is the policy that maximizes the expected return from each state. The optimal value function $V^*(s)$ gives the best expected return achievable from state s, and it satisfies the Bellman optimality equation:

$$V^*(s) = \max_{a \in \mathcal{A}} (R(s, \pi(s)) + \gamma \cdot \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s, \pi(s)) \cdot V^*(s_{t+1})).$$

This equation expresses the optimal value of a state as the maximum expected return over all possible actions.

The RL loop (see Figure 2.1a) proceeds through a series of steps as the agent interacts with the environment: The agent is initialized with a random policy or a predefined policy. Then the agent selects an action a_t based on its current policy π. It performs the action a_t, transitioning the environment to a new state s_{t+1}, and receives a reward r_t. Lastly, the agent updates its policy based on the observed rewards and new state and aims to improve its future performance. These steps are repeated for multiple time steps while gradually improving the policy.

A crucial challenge in RL is balancing the exploration-exploitation trade-off. Exploration involves searching unexplored areas of the solution space to potentially find better solutions, while exploitation focuses on refining known good solutions to improve performance. The dilemma arises from the need of the right balance: too much exploitation may lead to getting stuck in local optima, whereas excessive exploration can prevent the algorithm from converging to an optimal solution. Maintaining this balance is essential for the effective and efficient optimization of policies in RL [Fel24].

2.1.1. Policy Optimization

A key distinction in RL algorithms is whether the agent has access to a model of the environment. A model refers to a function that predicts state transitions and rewards in the environment. In model-free RL, the agent learns directly from interactions with the environment without any explicit model. In contrast, model-based RL involves learning or using a model of the environment's dynamics. The agent can plan and simulate actions using the learned model before taking actions in the environment. All in all, model-free RL simplifies the process by eliminating the need for a model, while model-based RL offers potential advantages in planning and exploration but faces the significant challenge of model learning and the risk of model bias.

This work focuses on model-free RL, specifically on Policy Optimization methods. An example which is also used in this work is Proximal Policy Optimization (see section 2.1.2). Policy Optimization methods explicitly represent a policy as $\pi_\theta(a|s)$ and aim to optimize the parameters θ either through direct gradient ascent on the performance objective $J(\pi_\theta)$ or by maximizing local approximations of it. Policy Optimization is usually performed on-policy, meaning that the updates are based only on data gathered while following the most recent policy iteration. This contrasts with off-policy methods, where the agent can update its policy using data collected from previous or different policies, thus allowing the agent to learn from a broader set of experiences. Additionally, policy optimization often involves learning an approximator $V_\phi(s)$ for the on-policy value function $V^\pi(s)$, which is used to determine how to update the policy effectively [Ach18].

2.1.2. Proximal Policy Optimization (PPO)

One state-of-the-art policy gradient method for training an agent's policy network is Proximal Policy Optimization (PPO). The intuition behind PPO lies in the idea of improving the training stability of the policy by limiting the change of the policy at each training epoch, i.e., avoiding having too large policy updates. Smaller updates during training are empirically more likely to converge to an optimal solution, whereas bigger updates may direct a policy in the wrong direction. Essentially, PPO is a network which approximates a policy function used by the agent to make decisions. To do so, the network uses a clip function to limit the size of policy updates, allowing larger step sizes without compromising the stability of the gradient ascent process [SWD+17]. The clipped surrogate objective function ([WHT20]) of state-action pair (s_t, a_t) is defined below:

$$L_t^{CLIP}(\theta) = \min(r_t(\theta, \theta_{old})A_t, \mathcal{F}^{CLIP}(r_t(\theta, \theta_{old}), \epsilon)A_t),$$

where θ and θ_{old} are the parameters of the new policy and the old policy. $r_t(\theta, \theta_{old}) \widehat{=} \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio and A_t is the estimated advantage value of $A^{\pi_{\theta_{old}}}(s_t, a_t)$. The clipping function $\mathcal{F}^{CLIP}$ is defined as:

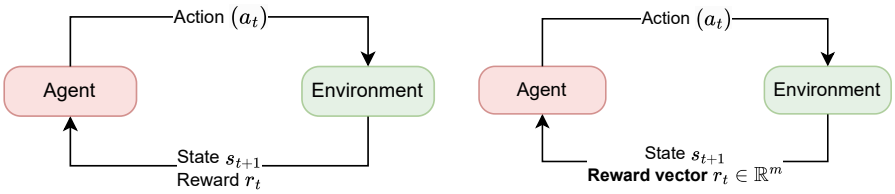
$$\mathcal{F}^{CLIP}(r_t(\theta), \epsilon) = \begin{cases} 1-\epsilon, & r_t(\theta) \leq 1-\epsilon \\ 1+\epsilon, & r_t(\theta) \geq 1+\epsilon \\ r_t(\theta), & else \end{cases}$$

where (1-ϵ, 1+ϵ) is called a *clipping range*, 0<ϵ<1 is the parameter. The PPO objective function aims to iteratively improve the new policy π_θ while keeping the updates within a controlled range. To summarise, the core idea behind PPO is that the policy update is weighted by the probability ratio $r_t(\theta, \theta_{old})$. If this ratio deviates too much from 1, i.e., the new policy differs too much from the old one, the update is constrained by the clipping function $\mathcal{F}^{CLIP}$. This ensures that while the policy improves, it does so within a controlled range resulting in the prevention of instability during training.

2.2. Multi-Objective Reinforcement Learning (MORL)

MORL is an extension of traditional RL that addresses decision-making problems where multiple objectives, potentially conflicting, need to be optimized simultaneously. Unlike traditional RL, which focuses on optimizing a single objective, MORL requires the agent to balance multiple objectives, each with its own reward function and potentially distinct discount factor. The key challenge in MORL is managing the trade-offs between these objectives, which often require the agent to learn a policy that balances conflicting goals. It is typically modeled using a Multi-Objective Markov Decision Process (MOMDP), which extends the traditional MDP by replacing the reward function with a vector of multiple reward functions $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^m$ with $R = [r_1, r_2, ...r_m]^T$ and discount factors $\gamma = [\gamma_1, ...\gamma_m]^T \in [0; 1]^m$, with the number of objectives m. Each reward function of R is of the form $r_t = R(s_t, a_t, s_{t+1})$, which aligns with the idea that in MORL, the agents receive a vector-valued reward rather than a scalar one. The interaction between the agent and the environment in MORL is illustrated in Figure 2.1, which compares the agent-environment loops in RL and MORL.

In contrast to traditional RL, where the agent focuses on optimizing a single objective, MORL requires the agent to balance multiple objectives that may conflict with each other. For instance, an agent may need to maximize both profit and sustainability, which could require trade-offs in the policies it learns. The challenge lies in finding the right balance between competing objectives, as optimizing one objective may negatively impact another. This leads to complex decision-making where the agent must determine how to prioritize and trade-off conflicting goals [HRB+22, VDB+11, ZDA23].



(a) RL agent-environment interaction loop. (b) MORL agent-environment interaction loop.

Figure 2.1.: Difference between RL and MORL. MORL extends traditional RL in making the agent receive a vectorial reward instead of a scalar [Fel24].

In traditional RL, the Bellman equation for a value function is used to describe the relationship between a state and the expected future rewards. Similarly, in MORL, it is extended to accommodate multiple objectives. The vector-valued Bellman equation in MORL is as follows:

$$V_i^{\pi}(s) = R_i(s, \pi(s)) + \gamma_i \cdot \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1}|s, \pi(s)) \cdot V_i^{\pi}(s_{t+1}), \text{ for } i = 1, 2, \ldots, m,$$

where $V_i^{\pi}(s)$ is the value function for the i-th objective, $R_i(s, \pi(s))$ is the reward function for the i-th objective, and γ_i is the discount factor for that objective. The Bellman equation describes how the value of a state is determined by the expected sum of rewards across all objectives, weighted by the respective discount factors.

The goal in MORL is to find a policy π^* that maximizes the value function across all objectives, which often involves finding an optimal trade-off between them. In practice, this may involve using a weighted sum of objectives or other techniques for selecting the best trade-off.

2.2.1. Pareto Fronts

In MORL, a policy that simultaneously optimizes all objectives does not exist. Thus, a set of non-dominated solutions is desired. While the goal of standard RL algorithms is a single optimal solution by optimizing a single reward function, MORL problems search for a set of solutions such that given any compromises between objectives, an optimal policy is in the set. This set is called a Pareto front.

Formally, policy π is dominated by policy π' when there is no objective under which π' is worse than π, i.e., $G_i^{\pi} \leq G_i^{\pi'}$ for $\forall i \in [1, 2, \ldots, n]$, with n being the number of objectives and G^{π} being the vector of expected return. A policy π is Pareto-optimal if and only if it is not dominated by any other policies [LPX+24].

The Pareto front consists of non-dominated solutions, representing the optimal set of trade-offs. A more diverse set of Pareto optimal solutions creates a more complete Pareto front, allowing to make flexible choices in MORL scenarios, rather than being restricted to a limited number of solutions. Therefore, MORL algorithms aim to provide a diverse set of Pareto optimal solutions [ZDA23]. In MORL, each point on the Pareto front corresponds to a policy's performance on different objectives (see Figure 2.2). However, in many complex real-world problems, it is often impractical to find the true Pareto front, so the goal is to approximate it with solutions that are both accurate and well-distributed along the front [VDB+11].

In some environments, the true Pareto front can be determined analytically or through exhaustive search, where the set of all non-dominated solutions is known beforehand. These true Pareto fronts serve as a benchmark for comparing the performance of different algorithms. However, many real-world environments, especially complex ones like those in *MuJoCo*, do not have a readily available true Pareto front. In such cases, finding

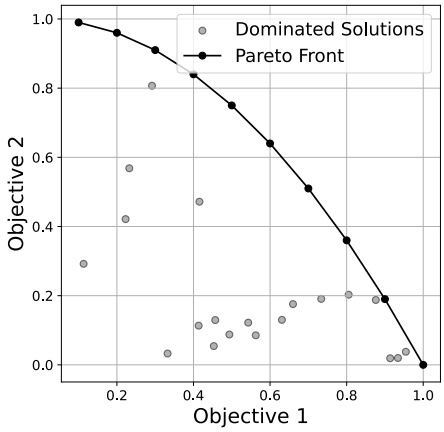


Figure 2.2.: Illustration of a Pareto front. The black points indicate solutions which form the Pareto front; all grey solutions are dominated by at least one member of the Pareto front.

the true Pareto front is often impractical due to the high dimensionality of the problem or the complexity of the underlying dynamics. Instead, the goal is to approximate the Pareto front as closely as possible and aim for a set of solutions that is both diverse and representative of the possible trade-offs between objectives. This is why, in environments like *MuJoCo*, the Pareto front is not a fixed or known quantity, and algorithms must rely on approximation methods to explore the space of possible solutions.

While just looking at the resulting front can give a first idea of the quality of the set of solutions, measuring the objective quality with actual metrics is a necessary step for comparing the fronts with each other (see section 5.2, in particular Figure 5.2).

2.3. Evolutionary Algorithms (EAs)

EAs are an optimization technique inspired by Darwin's theory of evolution by natural selection. The idea of applying evolutionary principles to computational problems originated in the 1950s and 1960s, when researchers where looking for biologically inspired methods to solve complex optimization tasks. The approach was formalized later and demonstrated how populations of candidate solutions could evolve over time using selection, crossover, and mutation. Since then, EAs have been widely adopted in computer science for solving problems in domains such as machine learning, robotics, and multi-objective optimization [RV18].

These algorithms iteratively evolve a population of candidate solutions. In each iteration, referred to as a generation, the EA evaluates the fitness of solutions within a given environment. Fitness is a measure of how well a candidate solution performs with respect to the given optimization objective. The definition of fitness depends on the problem that is to be solved. For example, in a RL context, fitness could be the total cumulative reward an agent receives over multiple episodes. In a multi-objective setting, fitness may involve a trade-off between multiple competing objectives, such as maximizing efficiency

while minimizing cost. Once the fitness of all solutions is determined, selection mechanisms favor better-performing individuals and allow them to reproduce and pass their characteristics to the next generation. This reproduction process often includes genetic operations (explained below), which introduce variation into the population. Over multiple generations, solutions evolve towards higher fitness, ideally leading to optimal or near-optimal solutions for the given problem.

The evolution of solutions in EAs is driven by three core genetic operators: mutation, recombination (or crossover), and selection. The selection operator plays a crucial role when determining which solutions from the current population are — with respect to a predefined fitness function — fit enough to be carried over to the next generation. Fitness is relative; higher-fitness solutions are more likely to be selected, but lower-fitness solutions can still be chosen through mechanisms such as roulette wheel selection or tournament selection. The goal is to ensure diversity while guiding the population toward better solutions. Selection is responsible for retaining the superior solutions and therefore influencing the overall improvement of the population. Crossover, also known as recombination, is the process by which two parent solutions are combined to produce new offspring ("child"). Crossover blends features from both parents with the hope of leading to superior solutions. Typically, crossover produces one or more offspring by exchanging genetic material between the parents at one or more points in their structure. This mechanism imitates biological reproduction and is fundamental in exploring the search space effectively. Finally, mutation introduces random changes to an existing solution. This helps to maintain genetic diversity within the population and prevents premature convergence to suboptimal solutions. They all share the goal of producing more optimal solutions in subsequent generations [MSG99].

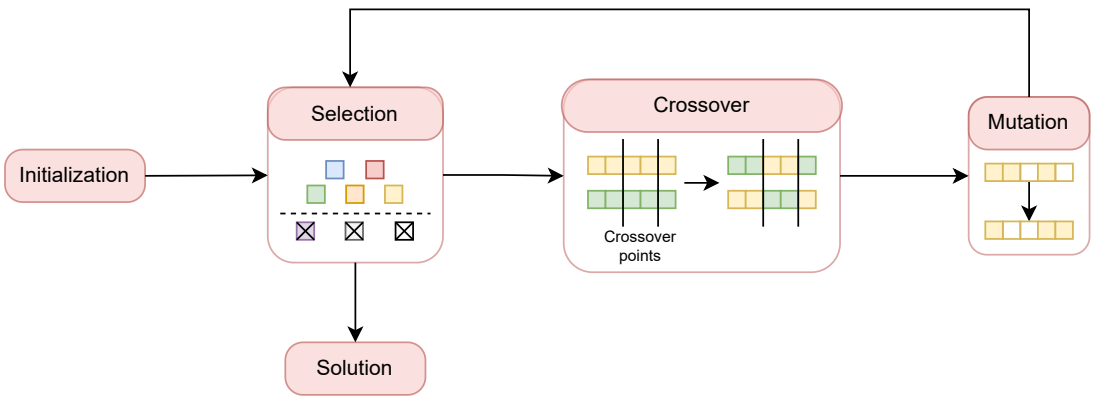


Figure 2.3.: Workflow of an Evolutionary Algorithm.

The workflow of an EA, represented in Figure 2.3, demonstrates how these operators interact to evolve solutions. It starts with an inital population, often generated randomly. Selection favors fitter individuals. Through crossover and mutation, new individuals are created and introduce variability and enable the search for optimal solutions. As generations progress, the population of solutions improves and moves closer to the (nearly) best solutions. While doing so, the algorithm explores different parts of the solution space

and avoids getting stuck on less effective solutions.

In summary, EAs are a powerful and flexible optimization method inspired by natural evolution to explore complex search spaces. By iteratively applying selection (favouring fitter solutions), crossover (combining features of parents), and mutation (introducing diversity), EAs evolve a population of policies while effectively balance exploration and exploitation. EAs are highly adaptable by allowing for customization through fitness functions and evolutionary operators. They are particularly useful for complex optimization problems, such as multi-objective tasks. Their ability to maintain diversity in the population helps avoid local optima; thus making them effective in solving challenging problems across various domains.

3. Related Work

MORL has been approached from various perspectives, with different algorithms focusing on balancing exploration, exploitation, and the generation of diverse policies along the Pareto front. This section provides an overview of key MORL methodologies relevant to this work. First, existing MORL algorithms are categorized to establish a broader context. Then, Evolutionary Reinforcement Learning (ERL) and Prediction-Guided MORL (PGMORL) are introduced, which serve as the foundation for the novel approach proposed in this work. Additionally, Concave-Augmented Pareto Q-Learning (CAPQL) and Generalized Policy Improvement (GPI) are described, as these methods serve as strong baselines against which the developed algorithm is later evaluated.

3.1. Classification of MORL Algorithms

In general, there are two main approaches when dealing with multi-objective problems: (1) single-policy and (2) multi-policy approaches. One of the simplest approaches to solving multi-objective problems is through the use of a scalarization function, which converts the multi-objective problem into a single-objective one. Each run produces only a single solution. To obtain a diverse set of trade-off solutions, multiple parameterized scalarization functions can be applied, and their results combined. Despite this, when it comes to the mapping from the weight space to the objective space, it can be challenging to define appropriate weights to ensure good coverage of the Pareto front. In contrast, multi-policy algorithms aim to identify a set of optimal solutions in a single run, rather than focusing on one solution at a time. Examples of such algorithms include evolutionary multi-objective methods which use a population-based search to evolve a diverse set of multi-objective solutions. Evolutionary multi-objective algorithms are widely regarded as some of the most effective techniques for dealing with complex multi-objective optimization problems [VMN14]. As a result, multi-policy approaches, particularly those based on EAs, are widely used in fields like RL, where multiple objectives (e.g., reward maximization, safety, and effectiveness) must be optimized simultaneously. Therefore, the next four sections outline algorithms which are also explored further within this work.

3.2. Evolutionary Reinforcement Learning (ERL)

Khadka & Tumer (2018) [KT18] introduced Evolutionary Reinforcement Learning (ERL), a hybrid approach that combines the strengths of Evolutionary Algorithms (EAs) and Deep Reinforcement Learning (DRL). Their work addresses three key challenges in DRL by (1) using a fitness metric based on cumulative rewards to handle sparse or delayed rewards, (2) maintaining a diverse policy population to enhance exploration in parameter and action space, and (3) ensuring robust and stable training through population-based

methods, reducing sensitivity to hyperparameters and convergence issues.

ERL integrates the exploratory capabilities of EAs with the learning efficiency of gradient-based DRL methods. It utilizes an evolutionary framework to generate diverse experiences for an RL agent while leveraging gradient information to refine the evolutionary process. The approach is adaptable to any off-policy reinforcement learning algorithm with an actor-critic architecture, though it was initially implemented using Deep Deterministic Policy Gradient (DDPG).

As shown in Figure 3.1, a population of actor networks is initialized with random weights. A separate RL-specific actor network and a critic network are also initialized. Each actor in the population interacts with the environment for a full episode. The fitness of each actor is calculated based on the cumulative reward obtained during the episode. The actors with the highest fitness scores are retained as elites and remain unchanged by mutation. Evolutionary operators, such as selection, mutation, and crossover, are applied to generate the next generation of actors. During interactions with the environment, each actor's experiences $\langle s_t, a_t, s_{t+1}, r_t \rangle$ are stored in a replay buffer. The critic samples from this buffer to update its parameters using gradient descent. The RL-specific actor is trained using these updates and periodically injected back into the population, introducing gradient-based improvements into the evolutionary process.

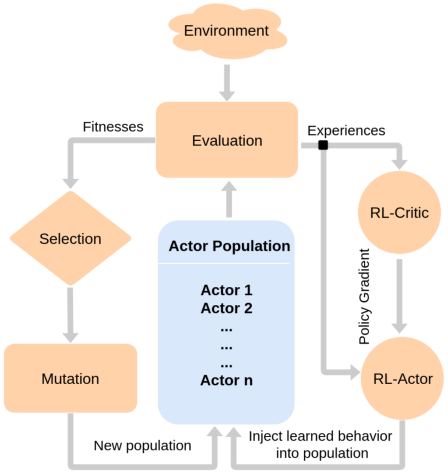


Figure 3.1.: Workflow of ERL. Incorporation of population-based learning with gradient-based optimization [KT18].

ERL leverages the complementary strengths of EAs and DRL. By combining the population-based exploration of EAs with the gradient-driven optimization of DRL, ERL provides a robust framework for solving complex RL problems. The evolutionary component ensures diversity and effective exploration, while the gradient-based RL component facilitates efficient learning from the generated experiences. This synergy allows ERL to address the limitations of conventional DRL methods, such as poor exploration and sensitivity to hyperparameters, while enhancing overall performance and stability.

Besides standard evolutionary operators like mutation and crossover, the authors suggest to incorporate more complex evolutionary sub-mechanisms, such as adaptive exploration noises or explicit diversity maintenance techniques. These enhancements could further refine the evolutionary process and allow ERL to better navigate complex environments.

3.3. Prediction-Guided MORL (PGMORL)

PGMORL, introduced by Xu et al. (2024) [XTM+20], is a population-based MORL algorithm that combines policy gradient methods with evolutionary strategies to optimize policies across multiple objectives. The method is designed to balance exploration and exploitation while improving the Pareto front approximation by leveraging prediction models and population-based optimization.

PGMORL integrates multiple key components to optimize policies across objectives effectively. It begins with a population of PPO agents [SWD+17], each associated with a performance metric and a weight vector representing trade-offs among objectives. A prediction-guided optimization process, driven by a model trained on historical policy data, predicts potential improvements for each policy along different optimization directions and selects promising policy-weight pairs to improve the quality of the Pareto front effectively. The whole algorithm and its stages can be seen in Figure 3.2.

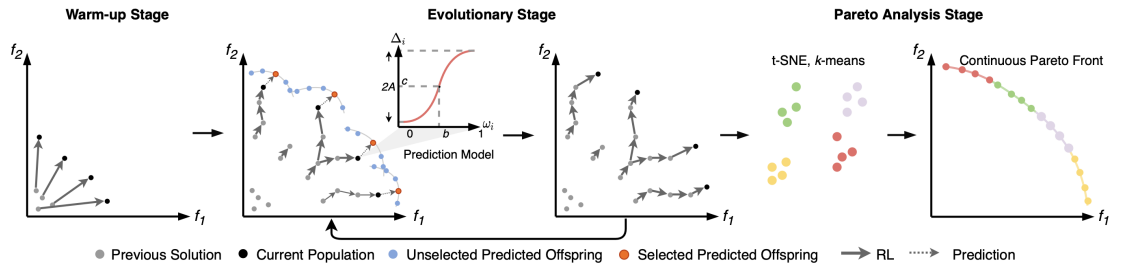


Figure 3.2.: Overview of PGMORL. *Warm-Up*: optimize n initial policies with different weights. *Evolutionary*: use a prediction model to select and optimize the best n policy-weight pairs to update the population and the model. *Pareto Analysis*: identify policy families and create a continuous Pareto front representation [XTM+20].

The algorithm starts with a warm-up stage, where n random policies are optimized using Multi-Objective Policy Gradient (MOPG) methods to get the initial policies out of the low-performance regions. In the evolutionary stage, selected policies are optimized in parallel using MOPG, producing offspring that update the population. To maintain performance and diversity, a performance buffer strategy is employed, while non-dominated solutions are stored in a Pareto archive. Instead of training all policies equally, the algorithm focuses on specific policies with the help of a prediction model. In this part of PGMORL, weight vectors are assigned, which guide the optimization through a tournament-style selection of policies and weight pairs most likely to improve

the Pareto front approximation. The weight vectors refer to parameters of these policies and are being used to evaluate them. In other words, during training the prediction model guides the decision on which policy to prioritize and therefore will make the greatest improvement along the Pareto front. The predictions are based on historical policy evaluations and associated weight vectors. Finally, the algorithm outputs a set of policies representing the Pareto front.

PGMORL makes no use of classic EAs where genetic operations like recombination, mutation, and selection are applied to evolve a population of policies. It primarily uses policy gradient updates and gradient-based methods rather than explicit evolutionary operations. Basically, PGMORL replaces EAs by RL but keeps the idea to evolve the Pareto front by weighted-sum scalarization or performance buffer for example. It is important to highlight that PGMORL continues to struggle with the issue of long-term local minima. The trade-off of spending more time on the local minima area in order to reach better performance, or spending time on optimizing other regions of the Pareto front, is still present.

3.4. Concave-Augmented Pareto Q-Learning (CAPQL)

Concave-Augmented Pareto Q-Learning (CAPQL) and Generalized Policy Improvement (GPI) (3.5) are described in the next two sections because besides PGMORL they are later compared against the algorithm developed in this thesis. Both methods aim to solve MORL problems by optimizing policies over multiple objectives, but they take different approaches in balancing exploration, exploitation, and Pareto front construction.

CAPQL extends Pareto Q-Learning by incorporating concave utility functions to guide the policy selection. A concave utility function in CAPQL combines multiple objectives (rewards) into a single value. As one objective improves, the additional benefit to the overall utility decreases. This ensures that CAPQL favors solutions that balance trade-offs between objectives rather than overly optimizing one objective at the expense of others. In essence, it guides the policy selection toward more balanced, well-rounded solutions. The key idea is to favor solutions that balance trade-offs between objectives rather than simply maximize individual rewards. It does so by guiding the action selection toward policies that optimize concave functions of the reward vector.

Instead of maintaining a single Q-value per state-action pair, CAPQL maintains a set of Q-vectors, where each Q-vector represents a different trade-off across multiple objectives. A concave function $U(Q)$ is applied to the Q-values to emphasize solutions that provide a more balanced improvement across objectives rather than extreme trade-offs. This helps to avoid solutions that are optimal for one objective but not ideal for others. CAPQL selects actions by maximizing the concave-augmented Q-values. This ensures that selected actions balance multiple objectives more effectively than standard Pareto Q-learning, which only maintains non-dominated solutions. The learned Q-values are used to get policies that approximate the Pareto front [LHY23].

3.5. Generalized Policy Improvement (GPI)

GPI is a framework designed to combine the strengths of multiple policies, each trained for different objectives, to create a superior overall policy. Whereas other algorithms train a single policy for all objectives, GPI combines policies that are trained with different preferences or weights to achieve an overall improvement.

The agent maintains a set of policies, each optimized for a different scalarized objective function or a specific trade-off in the multi-objective space. The core idea of GPI is that, at each step, the agent does not rely on just one policy but considers all stored policies and selects the action that maximizes the Q-value from the best-performing policy. That is, for a set of policies $\pi_1, \pi_2, \ldots, \pi_n$, the improved policy is defined as:

$$\pi'(s) = \arg\max_a \max_i Q^{\pi^i}(s, a).$$

The formula ensures that the action taken at state s is the one that maximizes the Q-value, where the Q-value is taken across the set of policies $\pi_1, \pi_2, \ldots, \pi_n$. It chooses the action a from the policy that offers the best outcome in that particular state. Thus, GPI combines information from multiple policies to improve the agent's decision-making by selecting the action that provides the highest utility, regardless of which policy it comes from. This ensures that the best decision is selected across all available policies. When a policy improves upon the others in a particular scenario, its experience can be used to inform and refine other policies. To put it in a nutshell, GPI identifies the most promising objective to train on at each step and also identifies which previous experiences are most relevant when learning a policy for a particular agent preference [ABR+23].

4. Methodology

This section outlines the methodology used in the development and implementation of the proposed algorithm. It is divided into three key sections. First, the conceptual framework is introduced which provides a general overview of the algorithm. After, detailed explanations of the functionality and underlying technology of the two main contributions are given: (1) a policy selection mechanism and (2) a training framework incorporating evolutionary operators.

4.1. Conceptual Framework

Traditional MORL algorithms typically train a great number of policies but return only a single policy, independent of the selection method. This approach lacks the flexibility to adapt to varying objectives at inference time. PGMORL, while effective in MORL, does not fully leverage classic EAs, such as mutation, crossover, and selection, which are essential for exploring the policy space. This reliance on policy gradient methods limits the framework's ability to escape local minima and hinders its performance in complex environments. Furthermore, PGMORL struggles with the trade-off between refining policies in local minima and exploring other regions of the Pareto front. ERL, on the other hand, suggests incorporating more complex evolutionary sub-mechanisms, such as adaptive exploration noises and explicit diversity maintenance techniques.

The algorithm proposed in this work — EMOS — addresses these limitations by (1) returning a trained set of policies instead of a single policy, (2) forming a Pareto front with the help of a population-based approach (see Figure 4.1a), and (3) enabling the adaption of search strategies throughout training by dynamically adapting exploration (uncertainty) and exploitation (regret) weights (see Figure 4.1b). The resulting policies collectively form a Pareto front which provides flexibility to dynamically select a policy that best fits specific objectives during inference.

The algorithm developed within this work builds on ideas from ERL and concepts from PGMORL (see section 3) and especially widens PGMORL's methodologies to incorporate the evolutionary framework. Both PGMORL and EMOS share a common framework of warm-up and evolutionary stages. This two-stage approach is advantageous because it gives the algorithm the opportunity to get the initial policies out of the low-performance region first. A detailed walk-through of the stages and their results is provided later in this section. This will offer a clearer understanding of how each modification contributes to improving the performance of the EMOS framework in MORL tasks.

Another distinction is that while PGMORL has a prediction model to guide policy selection, EMOS simplifies this by directly using predefined heuristics such as regret and uncertainty to select policies during training. The choice of regret and uncertainty

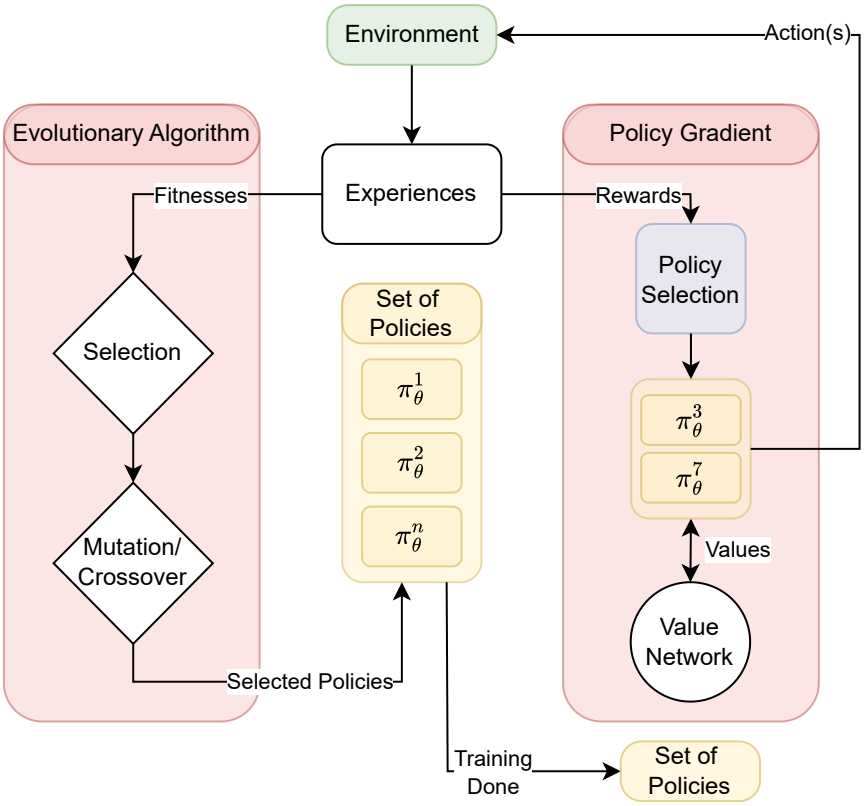
as heuristics for policy selection in EMOS stems from their intuitive alignment with exploration-exploitation trade-offs in multi-objective decision-making. Regret measures the difference between the best possible outcome and the outcome achieved by a given policy, which makes it a natural criterion for prioritizing policies that have the potential for improvement. Uncertainty, on the other hand, quantifies how confident the agent is in its expected returns and therefore helps to identify policies that require further exploration. By combining these two factors, EMOS aims to balance the need for selecting high-performing policies while still exploring areas of the Pareto front where information is limited. This approach makes EMOS computationally more efficient by avoiding the overhead of training an additional prediction model while still ensuring targeted improvements.

The goal of EMOS is to have a trained set of policies building a Pareto front, which allows for the selection of the most suitable policy based on the specific preference or goal at each time step. The flexibility to switch policies dynamically enables a more adaptable and responsive system. Besides selecting actions from the chosen policy during inference, the policy selection is also used during training, to choose which policy to update. As detailed in Figure 4.1b, rather than selecting policies based on predictions like PGMORL does, this algorithm selects policies based on the predefined heuristics.

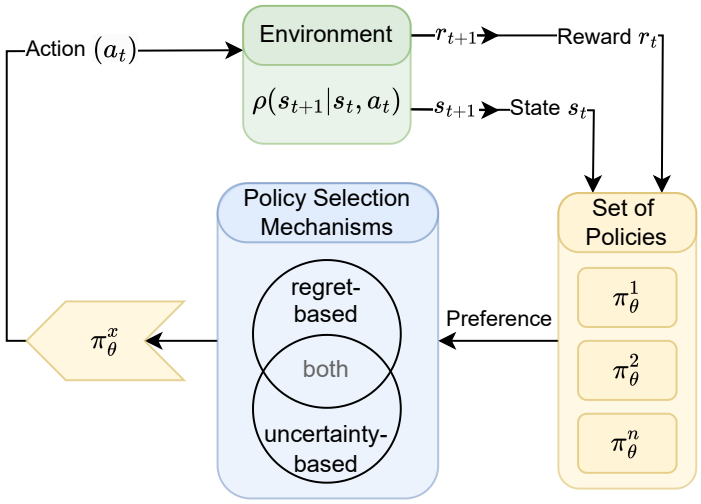
Among the other strategies, EMOS extends the idea of PGMORL in the sense that the evolutionary stage actually has evolutionary operators like mutation and crossover. The evolutionary strategy at the core of the conceptual framework makes the progressive improvement of policies easier through the mentioned operators.

Algorithm 1 Evolutionary Multi-Objective Selector Algorithm

Input: total timesteps T, reference point $r \in \mathbb{R}^2$, warm-up iterations W, evolutionary generations E
▷ **Warm-up Stage**
INITIALIZE random initial policies $\pi^i \in P$
for *iteration* $\leftarrow 1, 2, \ldots, W$ **do**
$P' \leftarrow \text{MOPG}(\pi^i)$
end for
▷ **Evolutionary Stage**
for *generation* $\leftarrow 1, 2, \ldots, E$ **do**
$\pi^* \leftarrow \text{PolicySelection}(T, r, P, e_{\pi^i})$ (see Algorithm 2) ▷ e_{π^i} are the correspondent evaluations of π^i
$P' \leftarrow \text{MOPG}(\pi^*)$
$P_{evolved} \leftarrow \text{EA}(P')$ (see Algorithm 3)
end for
Output: final trained set of policies $\pi^i \in P$



(a) EMOS's training framework



(b) EMOS's policy selection mechanism

Figure 4.1.: Schematic overview of proposed algorithm highlighting the incorporation of evolutionary operators with gradient-based optimization including only selected policies.

Just like PGMORL, the EMOS algorithm operates in two main stages (see Algorithm 1):

The training begins with a warm-up stage, where a population of policies is initialized randomly. These policies are optimized using multi-objective PPO (MOPPO) over a predefined number of warm-up iterations. MOPPO, which is an adopted component from the morl-baselines repository ([FAN+24]), extends the standard PPO algorithm (see Section 2.1.2) and calculates separate gradients for each objective and aggregates them to update the policy parameters. Unlike single-objective PPO, where the policy is optimized based on a single reward signal, MOPPO maintains a balance across multiple, potentially conflicting, objectives by adjusting the gradient updates accordingly. By incorporating MOPPO into the warm-up stage, the initial population of policies is guided toward a well-distributed set of solutions. This provides a strong foundation for the subsequent evolutionary optimization phase. This stage establishes a strong baseline for the evolutionary stage.

Once the warm-up stage is complete, the algorithm transitions to the evolutionary stage, where evolutionary strategies evolve the population of policies over multiple generations. The process involves the selection of policies to update and the actual evolutionary operators. Each action taken by a policy in a given state in the environment comes with a reward, which contributes to the overall fitness score and is used to calculate regret and uncertainty. At the start of each generation, policies in the population are selected based on these heuristics and are then updated using MOPPO. The updated population undergoes evolutionary operations to maintain diversity and improve the overall performance. The training is done when the total timesteps are reached. The detailed implementations and underlying methodologies of the policy selection mechanism and the EA are described in the next two sections and can be seen in the Algorithms 2 and 3.

4.2. Policy Selection Mechanism

In MORL, selecting appropriate policies for updating is an essential component for efficient exploration and exploitation of the solution space. It makes the development of diverse and well-performing solutions secure while preventing redundant updates and improving sample efficiency. The challenge lies in balancing the trade-off between exploring new solutions and exploiting known good policies to optimize performance across multiple objectives. In the following, the dynamic policy selection mechanism of the EMOS algorithm is described, especially how the training process is guided based on regret and uncertainty, with an adaptive weighting strategy to balance exploration and exploitation over training time. Through extensive tuning (see section 6.1.2), it was determined that instead of updating the best policy at each step, this mechanism updates the most promising policy at each iteration. With "promising", the worst policy according to regret and uncertainty is meant, i.e., the policy with the highest regret and highest uncertainty is updated because there is the most potential for getting better after the policy gradient update. Illustrated in Algorithm 2, the process involves calculating regret and uncertainty scores for each policy and combining them into a single score using a dynamically adjusted weighting scheme.

The heuristics regret and uncertainty are calculated like this:

$$\begin{aligned} \text{regret } R(e_\pi, r) &= \|e_\pi - r\|, \\ \text{uncertainty } U(e_\pi) &= \text{Var}(e_\pi), \end{aligned}$$

where $r \in \mathbb{R}^2$ is a reference point defined by determining the lower limit of each objective in the environment (see section 5.2.1) and $e_\pi \in E$ is the correspondent evaluation of a policy π [Wil24]. Therefore, regret measures how far the current policy is from an ideal reference point, which guides exploitation by prioritizing policies that perform closer to the ideal. Uncertainty, on the other hand, is the variance between the different reward functions and therefore measures the variability in a policy's evaluation which reflects the confidence in its performance. A higher uncertainty suggests that the policy's potential is less predictable and gives the opportunity of further exploration.

A balance between improving known good policies (low regret) and exploring uncertain regions for potential gains (high uncertainty) suggests the introduction of dynamic adjustments that adapt over the course of training. This adaptive weighting ensures that training initially explores broadly but gradually prioritizes stability and convergence as learning progresses and thus prevents premature convergence while maximizing long-term performance improvements:

$$\begin{aligned} \text{exploration weight } \alpha &= 0.8 \cdot e^{\frac{-\text{global step}}{\text{total timesteps}}}, \\ \text{exploitation weight } \beta &= 1 - \alpha. \end{aligned}$$

The constant factor 0.8 can be adjusted to control how quickly the exploration weight decays, while simultaneously increasing the exploitation weight as it is defined as $(1-\alpha)$. Initially, exploration (uncertainty) is favored, but as the training progresses and the global step increases, the exploration weight decreases, resulting in the promotion of more exploitation (regret). Since exploration and exploitation must sum to 1, the exploitation weight is simply the complement of the exploration weight. A score for each policy is calculated by combining both regret and uncertainty based on the current phase of the training. The agent will focus more on reducing regret during exploitation and more on reducing uncertainty during exploration. The negative signs ensure that the score is minimized which means that lower values correspond to worse policies (i.e., high regret or uncertainty). This formulation makes the function convex, allowing to find the worst policy by minimizing the score (see Algorithm 2):

$$\text{score } S(\pi) = -R(e_\pi, r) \cdot \beta - U(e_\pi) \cdot \alpha,$$

where $r \in \mathbb{R}^2$ is a reference point (see Section 5.2.1) and $e_\pi \in E$ is the correspondent evaluation of a policy π.

The process continues as the policy is selected with the lowest score, i.e., the highest regret and uncertainty. The policy's fitness, which is used for further evolutionary oper-

ators, is assigned during the policy selection process primarily for practical reasons (see Algorithm 2). Since the algorithm is already iterating through all policies to compute regret, uncertainty, and the novelty score (which is included in the fitness score anyway), it makes sense to directly assign fitness values at this stage. This way, the need for further iterations through the policies in the later EA and redundant calculations are avoided. While the calculation of fitness takes place in the selection phase, its detailed description is included in the EA part (see section 4.3), as it is a core component of the EA framework.

Algorithm 2 Policy Selection Mechanism

Input: total timesteps T, reference point $r \in \mathbb{R}^2$, set of policies π^i with $i \in \{1, 2, ..., |P|\}$, policy evaluations $e_\pi \in E$
INITIALIZE candidate weights ▷ see explanation of candidate weights here
INITIALIZE empty set of selected policies
for each $\pi^i \in P$ **do**
COMPUTE regret $\leftarrow R(e_{\pi^i}, r)$
COMPUTE uncertainty $\leftarrow U(e_{\pi^i})$
COMPUTE dynamic exploration α and exploitation β weights
COMPUTE $S(\pi^i) \leftarrow -R(e_{\pi^i}, r) \cdot \beta - U(e_{\pi^i}) \cdot \alpha$
end for
SELECT $\pi^* \leftarrow \min(S(\pi^i)) \in P$
COMPUTE novelty$(\pi^*) \leftarrow \frac{1}{k} \sum_{i=1}^{k} d(\pi^*, \text{neighbors}_{\pi^*})$
COMPUTE dynamic novelty weight ω
DETERMINE fitness$(\pi^*) \leftarrow (1 - \omega) \cdot \sum(\text{e}_{\pi^*}) + \omega \cdot \text{novelty}(\pi^*)$
Output: selected policy π^* with assigned fitness(π^*)

Since each policy represents a neural network, the term "candidate weights" is taken over from PGMORL and refers to parameters of these policies that are being used to evaluate and select the policies. These are the policies that are being considered for selection.

The proposed dynamic policy selection mechanism balances exploration and exploitation by computing regret and uncertainty for each policy, weighting them adaptively, and selecting the policy with the highest potential for improvement; it further incorporates a novelty measure to ensure diverse solutions while assigning a fitness score based on both performance and novelty of the respective policy. By doing so, it guides the training process toward an optimal set of solutions along the Pareto front. By emphasizing regret and uncertainty, this mechanism provides a framework for training agents in MORL environments.

4.3. Evolutionary Algorithm

In the context of RL, EAs are particularly useful for optimizing policies when traditional gradient-based methods might struggle. Since EAs do not rely on gradients, they can effectively handle complex, high-dimensional search spaces. This makes them well-suited for RL problems where the optimization landscape may be challenging due to the non-linearity and the potential presence of local optima.

The general workflow of an EA was already described in section 2.3. When bringing EAs into the multi-objective optimization context, they are particularly advantageous in balancing different objectives simultaneously. The population evolves over time to find solutions that best approximate the Pareto front. The combination of the EA's population-based exploration and the gradient-based approach with selected policies can lead to more efficient policy optimization across multiple objectives.

As recommended in the paper from Khadka & Tumer (2018) [KT18], more advanced evolutionary mechanisms beyond standard mutation and crossover should be incorporated to solve complex multi-objective optimization problems efficiently. Therefore, among other operators the evolutionary part of EMOS includes an adaptive exploration and exploitation noise and explicit diversity maintenance techniques.

EAs in MORL operate by evolving a population of policies over multiple generations, guided by selection, mutation, and crossover mechanisms. Specifically, in every evolutionary iteration, the selected policies are updated, guided by regret and uncertainty (see section 4.2). Following this, the population evolves through a series of steps: the top-performing policies are selected based on fitness — measured by a combination of the best evaluation scores and novelty (see the exact structure of fitness later in this section). The top policies serve as parents for crossover and produce the offspring that are then mutated. These top policies are also the elites and are automatically inserted into the next generation, without undergoing any changes like mutation. The weakest policies in the population are replaced to ensure continuous improvement. This evolutionary framework allows for the optimization of policies by leveraging the principles of natural selection, ensuring the gradual refinement of solutions over time [KT18]. Besides the simplified structure of population evolution shown in Algorithm 3, each component and operator is explained in more detail in the following.

The top-performing policies are selected for reproduction based on their fitness. Instead of only maximizing the cumulative rewards, the fitness function considers reward diversity across policies. It combines the evaluation score with a novelty measure. The novelty component encourages exploration of less populated or new regions in the objective space with the aim of a broad search for optimal solutions. Policies that contribute diverse rewards are favored, so it prevents the population from converging on a narrow set of solutions. The fitness of a policy is calculated during the policy selection process (see Algorithm 2), where the policy to update is chosen. The selected policy is directly assigned a fitness score by calculating its novelty. However, since fitness is an EA parameter, its detailed explanation is provided later in this section. During hyperparameter tuning (see section 6.1.2), it turned out that an adaptive novelty weight is beneficial in the sense of favoring novelty at the start of training and decrease the weight as the tranining progresses to exploit known good solutions by weighting the reward more.

Algorithm 3 Evolutionary Algorithm

Input: set of policies π^i with $i \in \{1, 2, ..., |P|\}$
SELECT parents
$\{\pi^1, \pi^2\} \leftarrow \text{argmax}_{\pi^i \in P} \text{fitness}(\pi^i)$ ▷ for fitness calculation see Algorithm 2
RECOMBINE parents
if random() > recombination rate **then**
return $\pi \in \{\pi^1, \pi^2\}$
end if
crossover points $i_1, i_2 \leftarrow \text{random}(1, \ldots, |\pi^1|)$ for two points ▷ multi-point crossover
$\pi_{child} \leftarrow [\pi^1[i]$ **if** $i == i_1$ **else** $\pi^2[i]$ **for** _ in range$(|\pi^1|)]$
MUTATE child
COMPUTE dynamic mutation rate m
$\pi_{mutated} \leftarrow [\theta + m \cdot N(0, 1) \forall \theta \in \pi_{child}]$
REPLACE weakest individual
$\pi_{weak} \leftarrow \text{argmin}_{\pi^i \in P} \text{fitness}(\pi^i)$
update π_{weak} with π_{child}
ENSURE $\{\pi^1, \pi^2\} \in P_{evolved}$ ▷ elitism
Output: evolved set of policies π^i with $i \in \{1, 2, ..., |P_{evolved}|\}$

The formula used to decay the novelty weight over time is similar to the dynamic adjustments in exploration and exploitation when weighting regret and uncertainty (see section 4.2):

$$\text{novelty weight } \omega = 0.8 \cdot e^{\frac{-\text{global step}}{\text{total timesteps}}},$$

where the global step tracks how many steps have been completed in training, while total timesteps is the total number of training steps. The term $e^{\frac{-\text{global step}}{\text{total timesteps}}}$ smoothly decays the novelty weight during training. This means that early in training, the novelty is more important (higher), but with training gets less important and therefore decreases. The constant factor 0.8 can be adjusted to control how quickly the novelty weight decays. This allows for a more efficient exploration-exploitation trade-off. The novelty weight is then multiplied with the actual novelty score to determine its weight within the fitness score.

The novelty of a policy is calculated as the average Euclidean distance to its k-nearest neighbors in the population:

$$\text{novelty}(\pi) = \frac{1}{k} \sum_{i=1}^{k} d(\pi, \pi_i),$$

where π_i are the k-nearest neighbors of π in the population. $d(\pi, \pi_i)$ is the distance between π and π_i given by

$$d(\pi, \pi_i) = \sqrt{\sum_{j=1}^{n} (\pi_j - \pi_{i,j})^2},$$

where n is the number of objectives in the reward vector. To calculate the Euclidean distance between two policies, their performance across multiple objectives is compared by using their reward vectors. As illustrated in the formula above, the process involves subtracting the corresponding values of each objective from the two policies, squaring the differences, and summing these squared differences across all objectives. Finally, the square root of this sum is taken to obtain the Euclidean distance. The resulting scalar value represents how "far apart" the two policies are in the reward space, which is essentially a multi-dimensional space where each dimension corresponds to an objective (in the case of the environments used in this work, two). If two policies have similar performance, i.e., their values for each objective are close to each other, the Euclidean distance will be small. On the other hand, if the policies perform very differently, the distance between them will be larger. This allows for assessing the novelty of a new policy by measuring how far it is from the other policies in the population.

Together with the reward score, the fitness of a policy consists of:

$$\text{fitness}(\pi) = (1 - \omega) \cdot \sum(\text{e}_\pi) + \omega \cdot \text{novelty}(\pi),$$

where ω is the novelty weight (see above) and $e_\pi \in E$ is the correspondent evaluation of a policy π. This composition ensures that fitness is a weighted combination of reward evaluation and novelty while dynamically balancing exploration and exploitation throughout training.

Each individual in the population represents a deep neural network. Crossover combines the weights of two parent networks to create offspring and exchange knowledge between individuals to potentionally generate novel, higher-performing policies. The multi-point crossover function combines the genetic material from two parents by alternating between their parameters at two randomly selected crossover points. The recombination rate is the probability that crossover will be performed; if crossover is not done (based on the recombination rate), one of the parents is randomly selected with equal probability (50%) and returned as a clone. Like this, no bias is introduced toward either parent when skipping crossover. Since no new offspring is created, this mechanism helps preserve genetic diversity by maintaining variability from either parent.

Mutation introduces random perturbations to the weights (genes) of these neural networks. Since the offspring after applying crossover is a combination of the parents' traits, it might not be fully optimized, and mutation helps explore new possibilities for improvement. Therefore the child is being mutated after crossover. Instead of a static mutation rate, it turned out that an adaptive mutation rate based on fitness creates more diverse policies (see section 6.1.2). The mutation rate defines the degree to which mutation is applied to the offspring. The mutation rate is dynamic, adapting based on the fitness of

the child: A higher fitness leads to a smaller mutation rate which results in better policies being less mutated and worse policies being more mutated.

A policy is being mutated like this:

$$\text{mutation rate } m = \frac{\text{base mutation rate}}{(1 + \text{fitness}(\pi))},$$
$$\pi_{\text{mutated}} = \theta + m \cdot N(0, 1) \forall \theta \in \pi.$$

Including a base mutation rate of 0.05 in the adaptive mutation formula, identified during hyperparameter search (see section 6.1.2), ensures that the mutation dynamics remain grounded within a predefined range. The base mutation rate serves as the upper limit for mutations, ensuring that even at the early stages of training or for low-fitness policies, the changes introduced are not excessively large, which could destabilize the search process. As the fitness improves, the adaptive formula reduces the mutation rate proportionally to exploit promising policies. This approach ensures that the algorithm adapts effectively to the different phases of training: starting with broad exploration when fitness is low and transitioning to fine-tuning as fitness improves.

Finally, the weakest policies in the current population are replaced with the new offspring generated through the operators mentioned above. Elitism refers to a selection mechanism in EAs and ensures that the best policies (elites) from the previous generation are preserved without modification to guarantee that the highest-quality solutions are carried over to the next generation. By preserving these elite policies, the algorithm maintains a strong foundation of well-performing solutions while still allowing diversity to be introduced through the offspring.

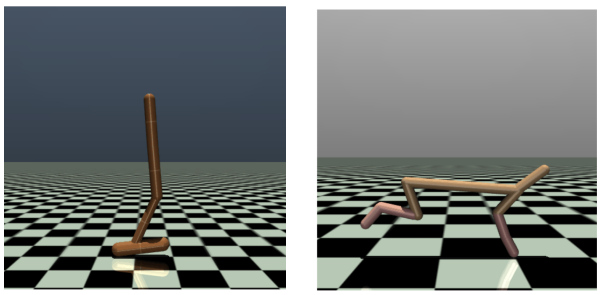
The interaction of these operators guarantees a balanced evolutionary process. A hybrid approach that combines these evolutionary operators and policy gradient methods can be effective. EAs can ensure diversity by selecting policies exploring various reward combinations, while the policy gradient can enhance the performance of policies with high regret or uncertainty, improving their outcomes across different reward dimensions.

5. Implementation

This section is about the practical aspects of developing and evaluating the proposed algorithm. It begins by outlining the environments utilized to test the algorithm's effectiveness. Then, the foundation for performance evaluation is provided, by describing the key metrics used in MORL to measure the algorithm's success throughout the development.

5.1. Benchmarks

In order to evaluate the proposed algorithm and benchmark it against state-of-the-art algorithms, experiments have been realised with the help of the MO-Gymnasium API [AFT+22, FAN+24]. MO-Gymnasium is a library containing a set of MORL environments which follow the standard Gymnasium API [TKT+24], but return vectorized rewards instead and are therefore suitable for multi-objective problems. In contrast to simpler environments, the group of MuJoCo *(Multi-Joint dynamics with Contact)* [TET12] has shown to be ideal for benchmarking RL algorithms in complex, real-world scenarios. Thus, experiments in this work include the MuJoCo environments *mo-hopper-2d-v4* and *mo-halfcheetah-v4*, both well-known benchmark environments available in MO-Gymnasium [AFT+22, FAN+24]. The next two subsections provide detailed descriptions of these environments, with visualizations in Figure 5.1 illustrating the tasks and dynamics involved in each scenario.



(a) mo-hopper-2d-v4 (b) mo-halfcheetah-v4

Figure 5.1.: Visual representations of the MO-Gymnasium environments used in the experiments [AFT+22].

5.1.1. mo-hopper-2d-v4

The reward function $r(s_t, a_t, s_{t+1}) \in \mathbb{R}^2$ of the multi-objective version of the Gymnasium's *Hopper-v4* environment is defined as:

$$r_1(s_t, a_t, s_{t+1}) = \text{forward speed on the x-axis},$$
$$r_2(s_t, a_t, s_{t+1}) = \text{jumping height on the z-axis}.$$

In this modified version, the agent must balance optimizing for its forward speed and jumping height [ETT12, FAN+24, XTM+20]. A three-dimensional reward function is available with the addition of energy cost as the third reward under the identifier *mo-hopper-v4*.

5.1.2. mo-halfcheetah-v4

This domain [FAN+24, Waw09, XTM+20] is a multi-objective version of the Gymnasium's *HalfCheetah-v4* environment. The multi-objective reward function $r(s_t, a_t, s_{t+1}) \in \mathbb{R}^2$ is defined as:

$$r_1(s_t, a_t, s_{t+1}) = \text{forward speed on the x-axis},$$
$$r_2(s_t, a_t, s_{t+1}) = \text{energy efficiency/control cost of the action}.$$

5.2. Evaluation Metrics

In contrast to single objective RL settings, where policies are evaluated in terms of their corresponding expected return, policies in MORL settings are evaluated using commonly-used MORL metrics computed based on Pareto fronts.

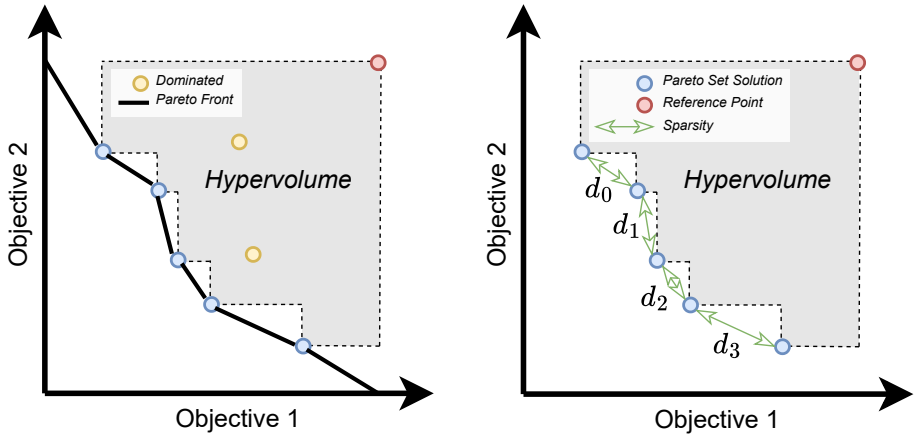
5.2.1. Hypervolume

The hypervolume metric is a widely used quality indicator in multi-objective optimization problems. As demonstrated in Figure 5.2a, it measures the volume of the solution space covered by a computed Pareto front with respect to a reference point. As the goal is to maximize the hypervolume, this reference point is usually defined by determining the lower limit of each objective in the environment, i.e., the worst return found across the whole training process [VMN14, DPT22, HRB+22]. A higher hypervolume value indicates better quality of the Pareto front and thus better solutions in terms of the objectives considered. The hypervolume provides a reliable reference for assessing the absolute performance of optimization algorithms. One of its critical features is its sensitivity to improvements in various characteristics of the Pareto front, such as accuracy, extent, and diversity. Any enhancement in these aspects will lead to a corresponding increase in the hypervolume value. Another important consideration is the timing of hypervolume evaluations during the training process. Results can vary depending on when the metric is assessed. To address this, the hypervolume metric is evaluated at periodic intervals, providing a more comprehensive understanding of algorithm performance over time [VDB+11].

The hypervolume is defined as:

$$HV(P, r) = \bigcup_{v^\pi \in P} \text{volume}(r, v^\pi),$$

where P is the Pareto front approximation in an m-dimensional objective space, and volume(r, v^π) is the volume of the hypercube spanned by the reference point $r \in \mathbb{R}^m$ and the vector v^π (vector for objective values for policy π) [FAN+24].



(a) Hypervolume (b) Sparsity

Figure 5.2.: Hypervolume and Sparsity Metrics in a 2-objective space. The shaded area, bounded by the Pareto front and the reference point, represents the region from which the hypervolume is derived. Sparsity is the average square distance d_i, where $i \in \{0, ..., 3\}$ between consecutive solutions in the Pareto front. In this case, $S = \frac{1}{4}(d_0^2 + d_1^2 + d_2^2 + d_3^2)$.

5.2.2. Expected Utility Metric (EUM)

This metric is used to evaluate how well a set of policies on the Pareto front performs across different weight configurations. MORL aims to optimize multiple conflicting objectives simultaneously, and as a result, the policies typically lie along the Pareto front. These policies are trade-offs between different objectives, and their effectiveness can vary depending on the relative importance of these objectives.

The EUM is defined as:

$$EU(P) = \mathbb{E}_{w \sim W}[\max_{v^\pi \in P} v^\pi \cdot w],$$

where W is the distribution of reward weights and w is a weight vector representing a specific trade-off between the objectives. w is sampled from the distribution W, which reflects different possible priorities for the objectives. The term $\max_{v^\pi \in P} v^\pi \cdot w$ selects the

maximum utility value obtained from applying the weight vector to each policy's objective vector. This ensures that the policy that performs best under a given weight configuration is selected for the expected utility calculation. v^{π} is the vector for objective values for policy π. The EUM specifically quantifies the utility of policies under various weight distributions. By sampling weights from a distribution W, the EUM evaluates the policies by applying the weights to the objective values of each policy in the Pareto front. The expected utility is then computed by taking the expected value of the weighted objectives [FAN+24, ZKR+15].

5.2.3. Sparsity

Sparsity shows how diverse a set of policies is in a given Pareto front by measuring the average distance between points in the front. This metric should ideally be low, indicating that solutions are more evenly distributed and closer together. As highlighted by Xu et al. (2020) [XTM+20], an optimal approximation of the Pareto front should aim to maximize the hypervolume, which reflects the coverage of the objective space by the solutions, while minimizing the sparsity metric. A low sparsity indicates that the solutions are well-distributed and closely spaced, ensuring that the Pareto front is densely populated and better represents the true trade-offs between objectives.

Sparsity is defined as:

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

where $\tilde{P}_j$ is a sorted list for the j-th objective values in P and $\tilde{P}_j(i)$ is the i-th value in this sorted list. An illustration of the metric is shown in Figure 5.2b.

5.2.4. Cardinality

Another metric used to evaluate the quality of the Pareto front is cardinality. It shows the number of non-dominated solutions in the approximate Pareto front obtained during optimization. This metric indicates the diversity of a solution set, representing how many unique solutions were found to balance multiple objectives. It applies: the higher the cardinality the better, because a higher value shows a more diverse Pareto front and thus more options to choose from.

6. Results and Discussion

This section provides an in-depth analysis of the findings from the development, evaluation, and comparison of the proposed algorithm with state-of-the-art algorithms. First, the focus is on a detailed evaluation of EMOS itself, including intermediate results where the algorithm is analyzed component by component, as well as hyperparameter tuning, which was crucial for optimizing its performance. Then the best-performing version of EMOS is benchmarked and its advantages highlighted in comparison to state-of-the-art algorithms.

6.1. Evolutionary Multi-Objective Selector (EMOS)

The detailed methods of the proposed algorithm are outlined in section 4, while the actual performance is presented and discussed in the following sections.

6.1.1. Intermediate Results

The intermediate results of the experiments performed to refine the EMOS algorithm correspond to modular components developed throughout this work. The primary objective is to refine the EA component to improve the overall performance in MORL tasks. Each component builds on the previous one, and by gradually adding more advanced methods, the identified limitations are addressed. The technical details of the resulting algorithm are already described in section 4, that is why this section outlines the modular components to build the final algorithm and especially the results connected with them.

The initial algorithm is a pre-version with basic EAs and policy selection based on static parameters. To address the problem of balancing exploration and exploitation even more, the first experiment investigates the impact of dynamic policy selection mechanisms. The hypothesis is that a static equal weighting between uncertainty and regret during policy selection does not dynamically adapt to different phases of training. As expressed in detail in section 4.2, regret measures how far the current policy is from an ideal reference point and uncertainty captures variability in the current evaluation. By balancing them during training, this variant showed positive outcomes regarding expected utility and hypervolume. The policy selection was able to better maximize the overall reward and improve the diversity of the solution set by covering a larger portion of the Pareto front. However, this version led to negative outcomes regarding cardinality and sparsity. This suggests that while the approach improves the trade-off between exploration and exploitation, it fails to generate a sufficiently large and diverse set of solutions.

This outcome prompted a shift in focus towards improving the evolutionary operators — mutation and crossover — which were initially seen as insufficient for effectively exploring

the solution space. The hypothesis is that by refining these operators, especially by incorporating adaptive mutation rates and multi-point crossover, the algorithm could generate a larger, more diverse set of policies that better explore the search space, thereby improving cardinality and reducing sparsity. Multi-point crossover is similar to single-point crossover with the only difference that multiple points are chosen for exchanging genetic material. This method can create a more diverse offspring, which may lead to a better exploration of the search space. As described in section 6.1.2, hyperparameter tuning helped identify key changes to the mutation rate that further enhanced these properties, leading to more promising outcomes in later experiments.

By utilizing dynamic policy selection and sophisticated evolutionary techniques, these experiments collectively seek to methodically create a more resilient EMOS framework that can perform well overall in multi-objective tasks. To particularly highlight the increased performance when adding complex EAs, this version is compared against the version with basic EAs. Both use dynamic policy selection mechanisms, i.e., they balance regret and uncertainty weights during training to optimize exploration and exploitation phases.

For a fair comparison, the earlier version with basic EAs is trained again with the optimized parameter values found during hyperparameter search (see section 6.1.2). Due to the exclusion of the complex EA parameters (adaptive mutation rate, novelty weight, and multi-point crossover) in the earlier version of EMOS, the optimized static parameters for mutation rate and novelty weight are used. These parameters were chosen to provide a fair comparison while drawing attention to the limitations of earlier versions of the algorithm. Figure 6.1 shows the results, while the shaded regions represent the 95% confidence intervals computed over multiple runs with eight different seeds. As the plots show, EMOS finds more diverse and well-spread solutions, as well as produces policies that align more effectively with the objectives. Regarding sparsity, EMOS is slightly worse than the version without the complex EAs in the way that solutions may be less evenly clustered. This makes sense since more complex EAs aim to find a more distributed set of policies — in this case, maybe too distributed. This results in policies being far away from each other and therefore resulting in a higher sparsity. However, the sparsity difference is small, hence EMOS is still overall superior, since hypervolume and expected utility are more critical in this scenario. These findings confirm that incorporating more sophisticated evolutionary strategies significantly increases the EMOS framework's ability to address MORL tasks.

6.1.2. Hyperparameter Search

For hyperparameter optimization, it is necessary to carefully select hyperparameters since they can significantly improve the performance of the algorithm, while suboptimal choices can lead to quite the opposite. As illustrated in Figure 6.2, hyperparameter search is definitely worth it, as the choice of parameter values (in this case the mutation rate, the novelty weight, the number of parents, and the recombination rate was sweeped) impacts the results drastically and shows the sensitivity of parameter changes. It is important to note, however, that sensitivity also depends on the environment, with some environments being more affected by parameter changes than others. The goal is to identify combinations that maximize the algorithm's performance and ensure robust results across

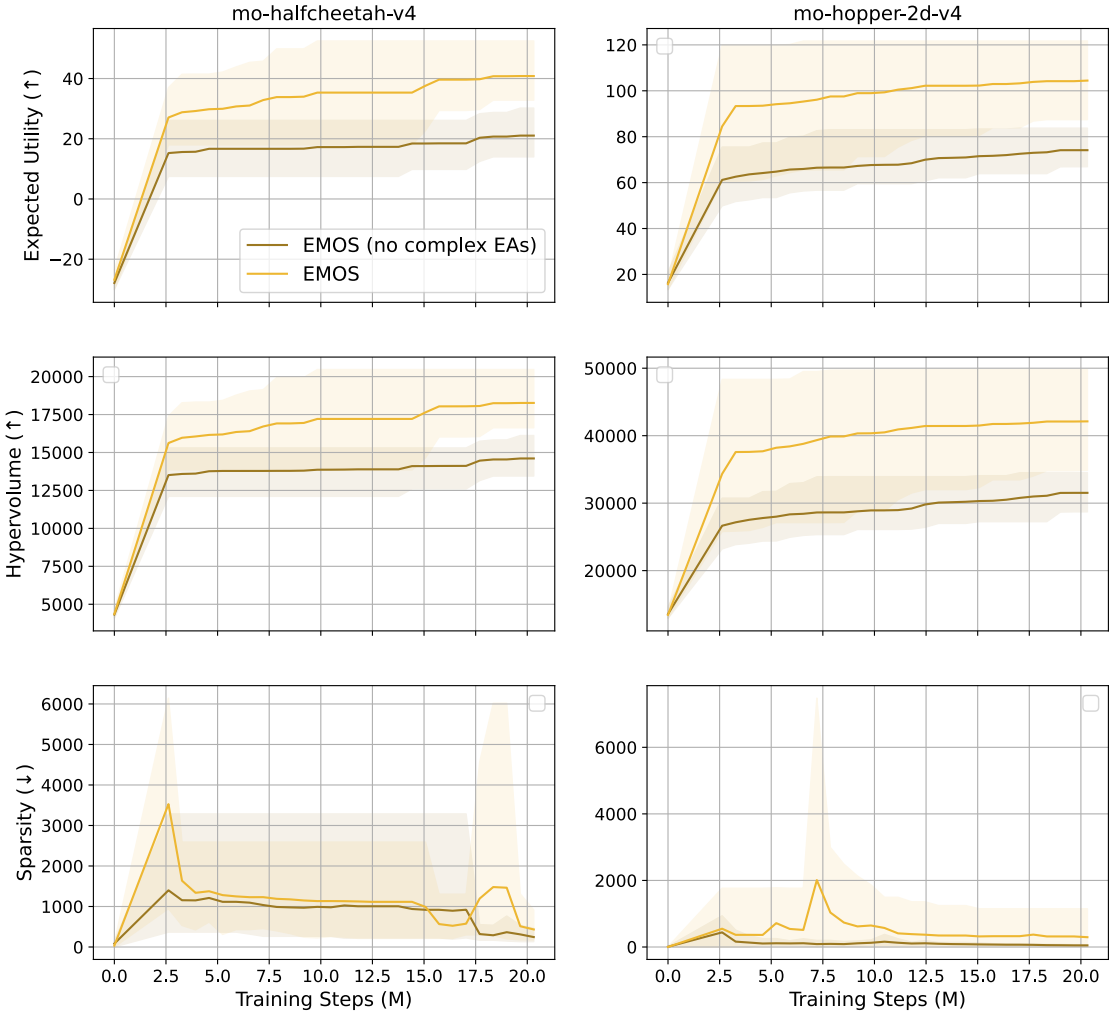
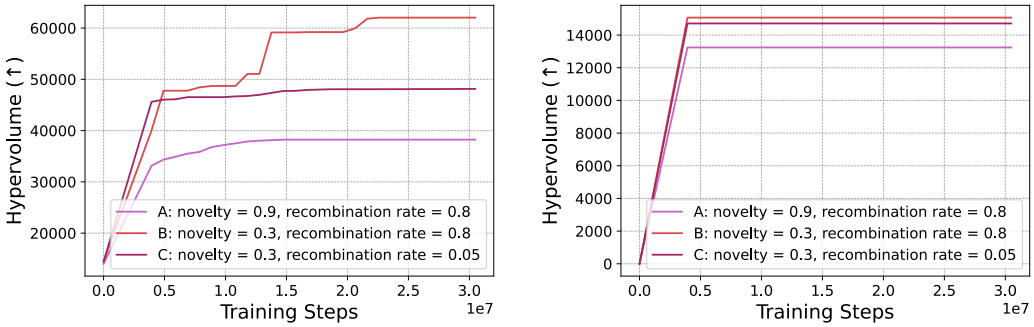


Figure 6.1.: Comparison of the intermediate version of EMOS without complex EAs with the final EMOS algorithm. The integration of complex EAs significantly increase the overall performance in both environments.

diverse environments. In this example, especially in the *mo-hopper-2d-v4* environment, the results vary significantly. The key parameters here include the novelty weight and the recombination rate, since they differ the most in these configurations. The combination of novelty = 0.3 and recombination rate = 0.8 (configuration B) achieves the highest performance in ensuring diversity without excessive exploration. A high recombination rate encourages a mixing of good policies and improves the overall learning efficiency. The lowest performance was reached with a combination of novelty = 0.9 and recombination rate = 0.8 (configuration A) because it over-explores new solutions rather than improving high-performing ones, making the approach too exploration-heavy. It is interesting to note that the respective configurations have the same order when it comes to performance, no matter in what environment the parameter values were tested. However, the *mo-hopper-2d-v4* environment seems to be more affected by parameter changes than the *mo-halfcheetah-v4* environment, according to the metrics used in this work.



(a) mo-hopper-2d-v4 (b) mo-halfcheetah-v4

Figure 6.2.: Impact of hyperparameters across different environments. Different configurations can significantly improve the algorithm's performance and therefore highlight the sensitivity of parameter changes.

This thesis focuses exclusively on optimizing the hyperparameters of the evolutionary part of the algoriothm, as the implementation of the EA builds the core component of this work. Other parameters, such as learning rate and the number of steps, are adopted directly from the PGMORL approach.

To find the optimal hyperparameter configurations for EMOS, a hybrid approach combining Bayesian Optimization (BO) and Grid Search is applied. BO focuses on sampling the most promising areas of the parameter space, based on prior evaluations, which might skip some combinations. Advantages include scalability, i.e., it is more suitable for large parameter spaces. Also, BO can more effectively explore continuous parameter spaces compared to the fixed grid points in grid search. If the setup is not properly configured though, it might focus too much on local optima and miss some global extremes. Grid Search, on the other hand, exhaustively tests all possible combinations of provided hyperparameter values within a specified range in order to find the best configuration. As the number of parameters increases, the number of combinations grows exponentially. Grid Search is inefficient for large spaces, because it may test many combinations that are not

optimal and lead to wasted computational resources [AL21].

This hyperparameter search includes four sweeps. In the first sweep, the following evolutionary operators are tested: mutation and recombination rate, novelty weight, and the number of parents for crossover. As mentioned in section 4.2, the mutation rate defines the degree to which mutation is applied to the offspring. Unlike the recombination rate, the mutation rate is NOT the probability of mutating a policy. The recombination (crossover) rate is the probability of combining traits from two policies to create a child. The diversity of the population is controlled by a novelty weight as a part of the fitness. It balances the reward and diversity. A weight closer to one encourages more diverse but possibly lower-performing policies to be selected. The agent might not learn anything anymore. A weight closer to zero may lead to a more homogenous population. Last, the number of parents serve as parents for crossover first of all, where, if there are more than two parents (e.g., four), random pairs of them are selected for recombination. These parents also indicate the elite fraction, i.e., the top-performing policies that are copied directly into the new generation. Elitism speeds up convergence and prevents the loss of good solutions. A fraction too high may reduce diversity and exploration [HAA+19]. BO is used here for narrowing down the search space and quickly identifying promising areas.

As it turned out after the first hyperparameter sweep, the parameter mutation rate shows promising results in both directions, small and high mutation rate. To avoid missing these global extremes, they are included again in the second sweep. According to the hypervolume and the expected utility, the following parameter values are selected:

- Mutation rate = 0.05, 0.8
- Novelty weight = 0.8
- Number of parents for crossover = 2
- Recombination rate = 0.8

The second sweep consists of testing the optimized algorithm against different sizes of the population size. Grid Search makes sure that after BO suggested promising parameter ranges, more fine granular values are found. To summarize, BO favors exploration and Grid Search favors exploitation by exploring the narrowed-down regions. The values above are now used for the second hyperparameter sweep. Here, the both extremes of the mutation rate, as well as different sizes of the population are tested. The population size is the number of policies trained in parallel, while larger sizes require more computational resources. As mentioned earlier, to prevent unnecessary testing of suboptimal values, Grid Search is used here with a limited amount of values. Results of the second sweep show that a mutation rate of 0.05 and a population size of 4 are more efficient in terms of hypervolume, expected utility, and overall diversity of the policies.

It remains to be determined whether adaptive parameters could improve performance in certain cases. As mentioned in section 6.1.1, it may be beneficial to use more complex EAs to explore the search space efficiently. Once optimal parameters for EMOS are found, it is important to consider whether parameters like mutation rate and novelty weight should be adapted during training. A static mutation rate does not adapt to different phases of training. Early on, exploration is crucial, but later, exploitation becomes

more important. Adapting the mutation rate dynamically ensures optimal performance throughout the training. Early in training, a high mutation rate is needed to explore the search space and avoid local minima. Later on, a smaller mutation rate helps exploit good solutions without deviating too much from the optimal solutions. Policies with higher fitness are assigned a lower mutation rate, resulting in less mutation and, therefore, less 'destruction' of already good solutions. Conversely, worse-performing policies receive higher mutation rates and get support in balancing exploration and exploitation. Similarly, the novelty weight determines the importance of new, diverse solutions versus rewarding fitness. At the start of training, favouring novelty encourages exploration. As the algorithm gets closer to optimal solutions, lowering the novelty weight and focusing more on fitness helps refine the solution. This dynamic shift ensures a better balance between exploring new solutions and exploiting known good ones. Thus, in the third sweep the optimal static parameters found earlier (mutation rate of 0.05 and novelty weight of 0.8) are compared against adaptive parameters. The results of the third sweep demonstrate that adaptive parameters outperformed the static ones. An explanation why can be found later in this section and detailed explanations on these parameters being adaptive are given in section 4.3.

Finally, the last aspect of this tuning is to compare the impact of updating the best policy or the worst policy in the policy gradient step. This is done to figure out whether prioritizing the refinement of high-performing policies (through elitism) or focusing on improving the underperforming ones leads to better overall algorithm performance. It turns out there was barely a difference in performance whether the best or the worst policy is updated throughout training. However, from a strategic standpoint, updating the worst policy was chosen as the final approach. This is because, by improving the underperforming policies, the algorithm encourages a more diverse range of solutions. Focusing on the worst-performing policies helps in enhancing the exploration aspect of the search, which is crucial during the later stages of training to ensure continued improvement and to prevent the algorithm from becoming stuck in local optima.

Hyperparameter	**Search Space**	**Selected Value**
Mutation rate	[0,1], adaptive	0.05, adaptive
Novelty weight	[0,1], adaptive	adaptive
Number of parents (crossover, elitism)	2, 4	2
Recombination rate	[0,1]	0.8
Population size	4, 6, 8, 16, 32, 64	4
Policy-Update strategy	worst, best	worst

Table 6.1.: Overview of tested parameters and final values for the EMOS algorithm.

In the end, this is the final configuration for the EMOS algorithm, which includes the parameters that were contributed within this work (for an overview and the range of tested values, see table 6.1):

- Base mutation rate of 0.05 with adaptive mutation
- Adaptive novelty weight

- Number of parents for crossover and elitism = 2
- Recombination rate = 0.8
- Population size = 4
- Update worst policy

After hyperparameter tuning, the selected parameters reflect a well-balanced approach between exploration and exploitation, optimized for solving MORL tasks efficiently. The following provides an interpretation and justification for each parameter:

The mutation rate adapts dynamically according to the fitness of the policies. Policies with lower fitness receive a higher mutation rate, while policies with higher fitness have a lower mutation rate. This ensures that the worse-performing policies undergo more significant changes to explore new areas of the solution space, while the better-performing policies are preserved and refined with smaller mutations to avoid "destroying" successful strategies. As a result, the mutation process balances exploration and exploitation based on the fitness of the policies throughout the training. The novelty weight adapts dynamically based on how far the training is (global steps) and shifts from high novelty in the early stages to encourage exploration, to lower novelty later to encourage exploitation. This makes sure that diverse solutions are explored in the beginning while the best candidates are refined as the training progresses. Going over to crossover, using two parents for recombination allows the algorithm to combine traits from two individuals, which can lead to more innovative solutions by inheriting strengths from both parents. These parents are also treated as elites, taken over by the next generation to preserve high-performing individuals. A high recombination rate of 0.8 indicates that most offspring are generated through recombination, which allows the algorithm to explore combinations of traits more extensively. A small population size of 4 ensures computational efficiency while maintaining sufficient exploration of the search space. Despite its size, the population achieves diverse coverage due to the algorithm's population-based approach. This is a key advantage of EMOS compared to other MORL algorithms like CAPQL/GPI, as it achieves similar performance measures with fewer policies (starts with population size of 4) and reduced computational load. One the one hand, continuously updating and retaining the best individuals from the population across generations aligns with elitism principles, where high-performing solutions are not lost due to stochastic processes like mutation or recombination. On the other hand, continuously updating the worst-performing policy ensures that weaker individuals improve over time rather than being discarded or replaced randomly. While elitism is inherently applied in the EA by ensuring that the two highest-performing parents are passed on to the next generation, EMOS further strengthens this process by specifically updating the worst-performing policy. Instead of being randomly replaced, these policies are given an opportunity to improve, while contributing to the overall diversity of the population. This balanced approach prevents the algorithm from focusing solely on the best policies, which might lead to premature convergence, and encourages ongoing improvement across the entire population, leading to more robust solutions.

The combination of a small population size, adaptive mechanisms, and a strong emphasis on diversity in general makes EMOS suitable for multi-objective optimization problems, where balancing diversity is essential.

6.2. Comparison with Baselines

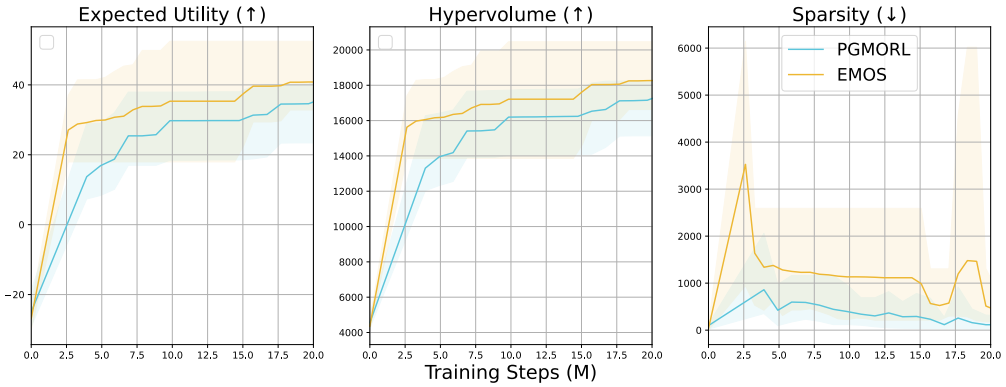
Now the overall evaluation of the proposed algorithm, EMOS, is presented by comparing it against PGMORL, CAPQL, and GPI. Since EMOS is an extension of PGMORL and shares a similar structural framework, their performance is separately analysed against each other first. Following this, EMOS is compared with CAPQL and GPI to measure its overall effectiveness across different MORL approaches.

To measure how well each algorithm approximates the Pareto front, the following metrics will continue to be used (compare section 5.2): Hypervolume, Expected Utility Metric, and Sparsity. In addition to these quantitative metrics, the visualized Pareto front is also analyzed later in this section to intuitively assess how well each algorithm approximates the trade-offs between objectives. However, it is important to be cautious when analyzing the Pareto front visually, as even a small number of policies could represent the most optimal solutions. Thus, the visual inspection must be done carefully, considering that fewer policies might still align closely with the ideal trade-offs.

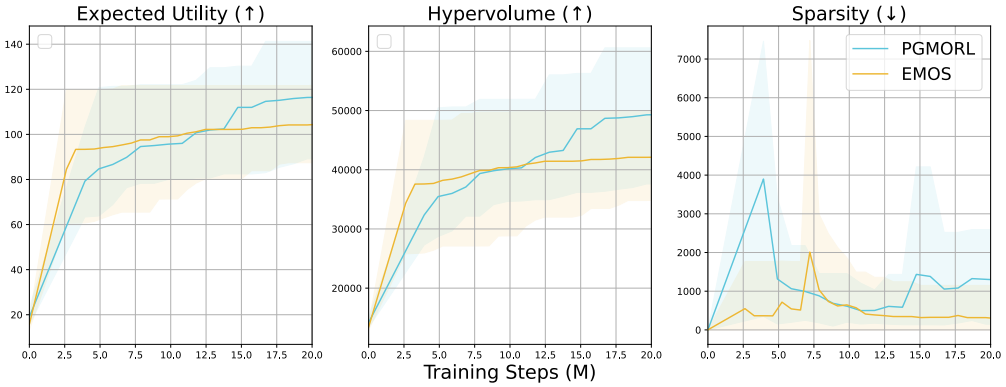
Regarding the experimental setup, the experiments were conducted in two environments: *mo-hopper-2d-v4* (see section 5.1.1) and *mo-halfcheetah-v4* (see section 5.1.2). Each algorithm was trained using the same eight different seeds per environment to ensure reproducibility and fairness, and thus eliminating variance due to random initialization and exploration differences. The shaded regions in the plots represent the 95% confidence intervals computed over multiple runs with the eight different seeds. Pareto fronts were constructed by selecting the front with the highest Expected Utility across all runs after the given training budget.

Considering PGMORL being the ancestor of EMOS, Figure 6.3 only shows those two algorithms in the spotlight. The figure compares EMOS and PGMORL in the known environments where each row shows the three key performance metrics expected utility (higher is better), hypervolume (higher is better), and sparsity (lower is better). On the one hand, EMOS and PGMORL show similar trends in expected utility and hypervolume across both environments. However, in *mo-hopper-2d-v4*, after EMOS having higher performance values in the beginning of training, PGMORL catches up eventually and achieves higher performance after approximately 10 million steps. This suggests that PGMORL is capable of learning strong policies over time, though it initially lags slightly behind EMOS. On the contrary, PGMORL does not catch up in the *mo-halfcheetah-v4* domain, resulting in EMOS being the overall better algorithm here and consistently outperforming PGMORL. EMOS may be more effective at covering the Pareto front, likely due to its evolutionary operators making diverse solutions possible.

Sparsity, on the other hand, reflects the distribution of solutions along the Pareto front, with lower values indicating more constant coverage. PGMORL maintains lower sparsity throughout training, especially in *mo-halfcheetah-v4*, whereas EMOS exhibits higher sparsity early on before stabilizing. One can argue that EMOS initially generates more scattered solutions due to its exploratory nature but gradually converges to a more structured set of policies. However, in *mo-hopper-2d-v4*, PGMORL's sparsity increases after about 10 million steps and gets higher than EMOS's sparsity. There could be a connec-



(a) mo-halfcheetah-v4



(b) mo-hopper-2d-v4

Figure 6.3.: Performance of EMOS compared to PGMORL in the *mo-halfcheetah-v4* domain and in the *mo-hopper-2d-v4* domain. EMOS inherits from PGMORL but replaces the prediction model by a policy selection mechanism and adds evolutionary operators.

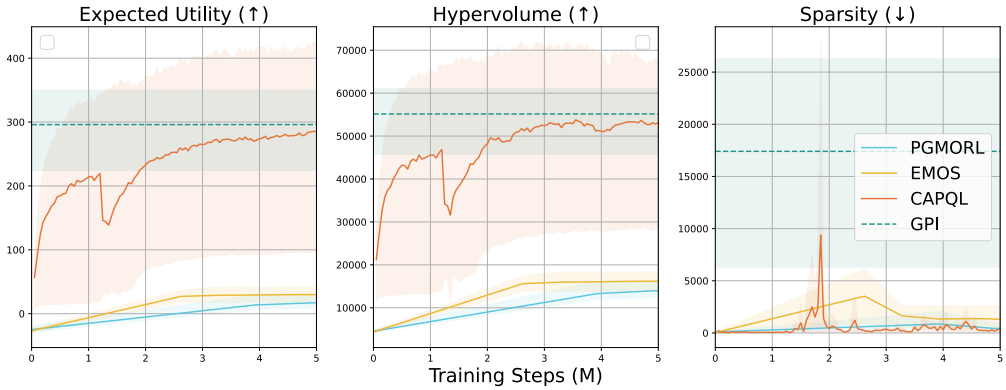
tion between PGMORL's increase in hypervolume and expected utility after 10 million steps and its simultaneous rise in sparsity. This gives the idea of a shift from broad exploration to exploiting fewer high-performing policies. Initially, PGMORL maintains a well-distributed Pareto front with low sparsity, but as training progresses, it discards suboptimal solutions which leads to gaps in the front. This indicates that PGMORL converges to fewer but stronger solutions, whereas EMOS retains more diverse policies due to its evolutionary mechanisms. As a result, EMOS maintains better overall performance in *mo-halfcheetah-v4*, while PGMORL catches up in *mo-hopper-2d-v4* but sacrifices diversity. The trade-off highlights PGMORL's tendency to focus on optimal rewards at the expense of maintaining a well-spread Pareto front.

Overall, EMOS demonstrates clear advantages over PGMORL in terms of hypervolume and expected utility, especially in *mo-halfcheetah-v4*. Moreover, EMOS achieves similar or even better results on average compared to PGMORL, while requiring fewer computational resources, as it does not learn a prediction model and works with fewer policies in the population. However, its slightly worse sparsity suggests that it explores more aggressively at the beginning, which may contribute to early instability. This trade-off highlights EMOS's strength in finding diverse, high-quality solutions, making it a strong alternative for solving problems in MORL domains.

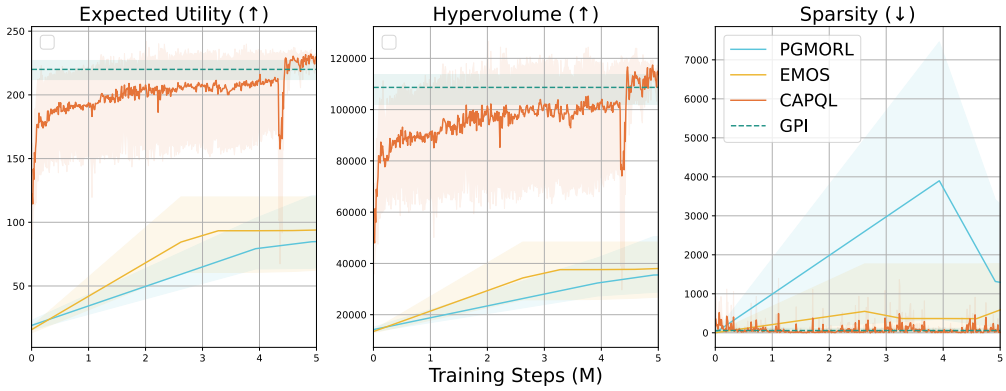
CAPQL and GPI were chosen for comparison against EMOS because they are compatible in terms of observation and action spaces and have been tested on the *MuJoCo* environments. If they are comparable regarding their individual approaches is discussed later in this section. The results from CAPQL from the morl-baselines repository ([FAN+24]) were reconstructed and then compared against the EMOS results. Note that the performance of GPI is illustrated as a dashed line with correspondent 95% confidence interval and represents the final values at the end of training for the respective metric. Since reconstructing GPI's results would have been computationally expensive, they were taken directly from the morl-baselines repository [FAN+24]. Additionally, previous studies have already conducted a comparative analysis of PGMORL, CAPQL, and GPI, which make them relevant baselines for evaluation [Fel24].

Figures 6.4a and 6.4b illustrate the comparison of EMOS with PGMORL, CAPQL, and GPI across the same performance metrics used throughout this thesis. The visualization was scaled to 5 million steps to better capture trends; however, a plot of the entire training process is included in the Appendix A.1 for completeness. Although EMOS performs better than PGMORL (see Figure 6.3), they both perform significantly worse than CAPQL and GPI (for a detailed explanation of these algorithms see section 3). More specifically:

CAPQL and GPI reach significantly higher expected utility values compared to EMOS and PGMORL, which generally suggests better policy optimization. In *mo-hopper-2d-v4*, CAPQL shows a notable outlier in hypervolume and expected utility around 4.4 million steps, where its performance briefly drops to a level close to EMOS and PGMORL. This could indicate instability or a temporary failure in policy updates. In contrast to the other algorithms, PGMORL shows an exceptionally high sparsity. This may indicate that the learned solutions are spread widely in the objective space but might not be well-



(a) mo-halfcheetah-v4



(b) mo-hopper-2d-v4

Figure 6.4.: Performance of EMOS compared to state-of-the-art MORL algorithms in the *mo-halfcheetah-v4* domain and in the *mo-hopper-2d-v4* domain. These algorithms were compatible in terms of the environment and they all share a multi-policy approach.

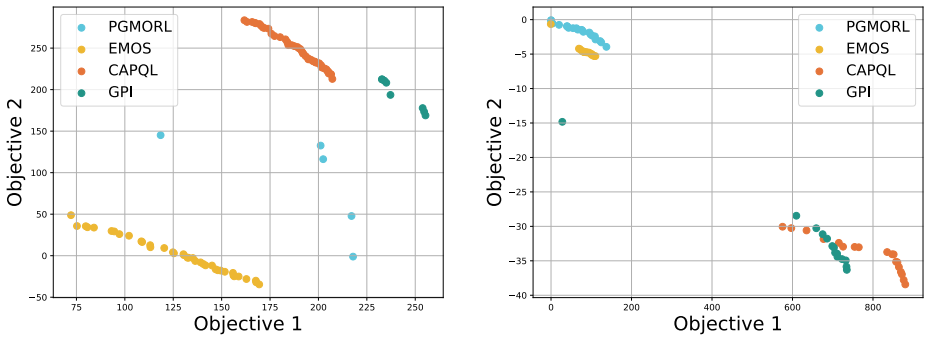
optimized at this point of training. When it comes to sparsity in the *mo-halfcheetah-v4* domain, GPI is extremely high including an immense confidence interval compared to the other algorithms.

The performance gap between PGMORL/EMOS and CAPQL/GPI can be attributed to two main factors: (1) algorithmic superiority of CAPQL and GPI and (2) methodological factors. GPI dynamically selects actions from past policies ([ABR+23]), whereas EMOS updates policies in isolation and therefore limits its learning efficiency. CAPQL and GPI operate off-policy which allows them to reuse past experiences, while EMOS follows an on-policy learning approach, updating policies based solely on their current performance. Furthermore, the CAPQL/GPI approach incorporates Q-learning, which provides more structured learning and knowledge sharing, whereas EMOS — besides policy gradient updates via PPO — relies on evolutionary operators like mutation and recombination. That possibly introduces more randomness and slows down convergence. A tuned population size of four in EMOS is relatively low and may have constrained performance. Additionally, EMOS selects policies based on regret and uncertainty, which may lead to suboptimal or overly specialized solutions instead of well-distributed ones. This is further discussed in section 7.2. While EMOS aims to explore policies through evolutionary mechanisms and exploit good policies through policy gradient updates, its stochastic nature requires more samples to reach high-quality solutions. In contrast, CAPQL and GPI systematically refine policies and that benefits from more efficient learning and sample reuse [LHY23].

All in all, the key differences why EMOS could have a worse performance outcome, are that it directly selects policies in isolation based on regret and uncertainty, which, while targeted, does not inherently promote well-distributed solutions. Quite the contrary: it may result in convergence to suboptimal or overly specialized policies. Further, EMOS actively evolves policies through selection, mutation, and recombination, but this process is more stochastic and may require significantly more samples to reach high-quality solutions. CAPQL and GPI benefit from structured learning, knowledge sharing, and sample efficiency, which make them significantly more effective.

In MORL settings, various metrics can be used to evaluate and compare Pareto fronts generated by different algorithms. This analysis focuses on identifying the best-performing front for each algorithm and environment by selecting the one with the highest expected utility across all runs. The resulting Pareto fronts are visualized in Figure 6.5. Before going into detail, the goal in MORL is not simply to generate a "nice-looking" front, but rather to generate solutions that maximize the chosen metrics. As long as the expected utility is better, the Pareto front will also be better, regardless of its visual appearance. Perhaps not all points from the Pareto front are necessary — it may be sufficient to focus on the key points that provide the best trade-offs, even if the full front appears less well-defined. This phenomenon can be demonstrated by analyzing specific runs where a high hypervolume does not correspond to a well-distributed Pareto front and vice versa, a low hypervolume may correspond to a visually appealing front (see Figure A.2).

In Figure 6.5a it can be seen that EMOS approximates a quite well-distributed Pareto front while CAPQL and GPI occupy a different part of the objective space. PGMORL



(a) mo-hopper-2d-v4 (b) mo-halfcheetah-v4

Figure 6.5.: Pareto fronts of MORL algorithms with focus on EMOS. They were constructed by identifying the front with the highest expected utility across all runs.

has sparser points, but like mentioned above, this does not mean a less effective approximation of the Pareto front. However, PGMORL appears to be located in a significantly different region of the objective space compared to the other algorithms. This suggests that PGMORL may be failing to explore regions where more optimal trade-offs between the objectives are achievable, particularly in the areas where CAPQL and GPI perform better. Therefore, despite the fact that PGMORL is still covering a portion of the Pareto front, its solutions may be suboptimal since they are not concentrated in the most promising regions for achieving better overall performance across objectives. In Figure 6.5b, the Pareto fronts of CAPQL and GPI are well-formed and dominate the lower-right region of the objective space, while EMOS and PGMORL appear less competitive in the *mo-halfcheetah-v4* environment. The Pareto front of EMOS looks quite promising, it is important to highlight though that the real question is whether these solutions effectively support the trade-offs between objectives, or if CAPQL and GPI might perform better in this regard. According to hypervolume and expected utility, CAPQL and GPI clearly outperform EMOS. The solutions found by EMOS, though well-distributed, may not be as optimal in terms of the trade-offs between objectives.

Overall, the different Pareto fronts of the respective algorithms are widely separated. This observation suggests that different algorithms explore distinct regions of the objective space, and thus leading to varying trade-offs between objectives. These observations highlight the diverse exploration strategies of the algorithms and the impact on the resulting Pareto fronts.

7. Conclusion

MORL is a powerful framework for solving problems that involve multiple conflicting objectives. Unlike traditional RL, where a single objective is optimized, MORL aims to find a set of solutions that represent optimal trade-offs between various objectives. This makes it particularly useful in complex real-world scenarios, where different objectives must be balanced. The key challenge in MORL is efficiently exploring the solution space to identify policies that achieve these trade-offs, while avoiding overly simplistic or sub-optimal solutions. Over the years, several methods, such as Pareto-based algorithms and evolutionary approaches, have been proposed to tackle this challenge, each with their strengths and weaknesses. The goal of this work is to introduce a new approach, EMOS, designed to address these challenges more effectively through targeted policy selection and evolutionary mechanisms. By leveraging these evolutionary strategies, EMOS aims to efficiently generate a diverse set of high-performing policies that adapt and improve over time. The experimental results demonstrate EMOS's potential, yet also reveal areas where improvements could be made, particularly in comparison to other established methods like CAPQL and GPI. This section discusses the implications of the findings, and outlines potential directions for future research to address current limitations and extend the proposed approach.

7.1. Implications of Findings

Before moving to the final results, the intermediate results and the hyperparameter search process that contributed to refining the EMOS algorithm are summarised. While EMOS produced more diverse solutions, it had slightly higher sparsity compared to the simpler EA version, but outperformed it in terms of hypervolume and expected utility. The hyperparameter search revealed that parameters like mutation rate and novelty weight significantly impacted performance, with sensitivity varying across environments. This work primarily focused on optimizing the evolutionary components, while other parameters were borrowed from the PGMORL approach.

EMOS demonstrated a stable and consistent approach to policy evolution along with a well-distributed Pareto front in certain environments, such as *mo-hopper-2d-v4*. Nonetheless, its performance lagged behind that of CAPQL and GPI, which explored broader regions of the objective space and offered superior trade-offs, as evidenced by the hypervolume and expected utility, which were much higher for CAPQL and GPI. EMOS's shortcomings were further confirmed in the comparison plots with CAPQL and GPI, which showed that EMOS failed to capture the full range of optimal solutions available to these other algorithms.

Despite EMOS's limitations in comparison to CAPQL and GPI, it still outperformed PGMORL in terms of hypervolume and expected utility, which are critical metrics for evaluating the quality and coverage of the Pareto front. PGMORL, while covering a portion of the Pareto front, showed a much sparser distribution of points. This implies that it was not able to explore the objective space as efficiently or effectively as EMOS. This is a significant finding, as it highlights that EMOS maintained a denser and more well-distributed set of solutions, which was better aligned with the goal of finding a diverse set of high-performing policies. This demonstrates that EMOS was more effective at exploring and exploiting the objective space, even if it did not match the broader exploration capabilities of CAPQL and GPI. EMOS still provided valuable insights into evolutionary approaches to MORL, and its performance could be significantly improved through targeted refinements.

In specific, to bring its performance closer to CAPQL and GPI, several key areas can be refined: EMOS currently performs best with a population size of four, which is quite small. A larger population could allow for better exploration and more robust policy evolution. However, it is worth noting that the small population size in EMOS may also have an advantage, as it can still achieve a well-distributed Pareto front despite the limited number of policies. This could imply that EMOS is more efficient in selecting diverse and high-performing solutions even with a small population. CAPQL benefits from structured knowledge sharing, while EMOS evolves policies in isolation. A mechanism that encourages diversity while still leveraging shared insights (e.g., clustering-based selection) could improve robustness by identifying distinct groups of policies with different characteristics.

By making these adjustments, EMOS could achieve more stable learning, better exploration-exploitation balance, and improved efficiency, making it more competitive with CAPQL and GPI.

7.2. Limitations and Future Work

Despite the promising results of this work, several challenges and limitations remain that could provide valuable directions for future research.

One key aspect to explore is the combination of regret and uncertainty in the policy selection mechanism. While this approach showed potential, further research is needed to determine whether other combinations of these factors could yield more meaningful results, particularly in different contexts and environments. No matter how, it is worth questioning whether regret and uncertainty are the most suitable metrics for policy selection in all situations. Depending on the specific goals of the task, other heuristics like greedy approaches or crowding distance might better guide policy evolution in certain environments.

The fact that depending on the algorithm the Pareto fronts are distributed in different regions of the objective space (see Figure 6.5), can be a valuable point for future work, as depending on the context, scenario, and specific goals, different algorithms can be applied to obtain the desired policy. The choice of the algorithm can directly influence

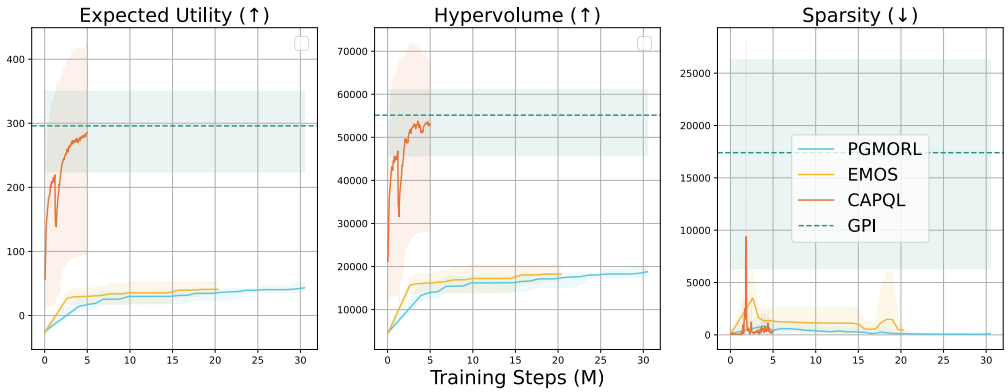
which region of the Pareto front the resulting policy will lie on, and allows for a more tailored solution based on the particular trade-offs one wants to prioritize. This is especially useful in real-world applications, where context-dependent solutions are often needed.

Additionally, the morl-baselines paper ([FAN+24]), where the implementation of PG-MORL is adapted and EMOS inherits from, did not involve exhaustive hyperparameter tuning of their benchmarks. The goal of their experiments was primarily to validate implementations and showcase algorithmic behavior across different evaluation metrics. The EMOS algorithm adopted these hyperparameter values though and only the added parameters like mutation rate or novelty weight were optimized in this work. Also, some hyperparameter choices were based on relatively small performance differences, and alternative settings might have improved the outcomes. Therefore, further improvements in EMOS could be achieved through more extensive hyperparameter optimization including the basic parameters which were taken over from the morl-baselines paper. On top of that, a more exhaustive evaluation with additional random seeds could strengthen the robustness of the findings. Applying EMOS to a wider range of environments, including those with more than two objectives, e.g., 3-dimensional, would provide deeper insights into its capabilities and limitations in the context of MORL. However, computational and time constraints limited the ability to conduct such extensive experiments.

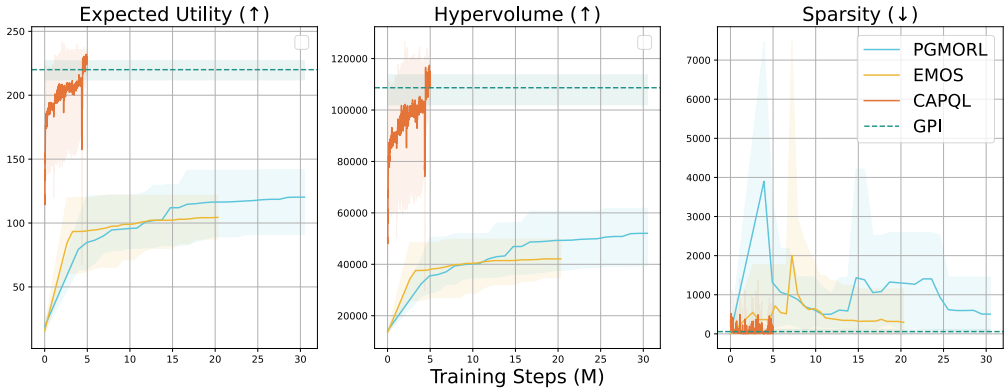
Another limitation is the adaptive novelty mechanism. While adaptive novelty was employed, it was primarily dependent on the number of global steps rather than dynamically responding to the actual diversity of the population. A more advanced approach could involve adjusting the novelty weight based on population stagnation: increasing it when the diversity is low to promote exploration and decreasing it when the diversity is high to enhance exploitation. This would allow for a more effective balance between exploration and exploitation throughout training. The same applies to the weighting of regret and uncertainty. These weights are currently adjusted through exploration and exploitation weights based on the stage of training but do not react to the actual status of the population.

As the field of MORL advances, it holds great potential to address increasingly complex real-world problems that require balancing multiple, often conflicting objectives. This makes MORL highly relevant for applications where decisions must consider several goals like safety, efficiency, and cost. By optimizing for multiple objectives simultaneously, MORL can lead to more effective solutions in these domains, where traditional single-objective approaches often fall short. Looking ahead, there is considerable opportunity to improve MORL algorithms by enhancing their scalability and efficiency, enabling them to handle more complex, high-dimensional problems. In the future, MORL could lead to more intelligent and adaptable systems capable of solving intricate multi-objective challenges across various industries.

A. Plots and Visualizations

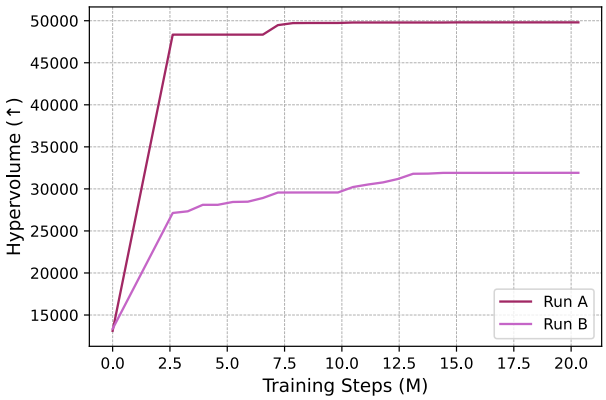


(a) mo-halfcheetah-v4

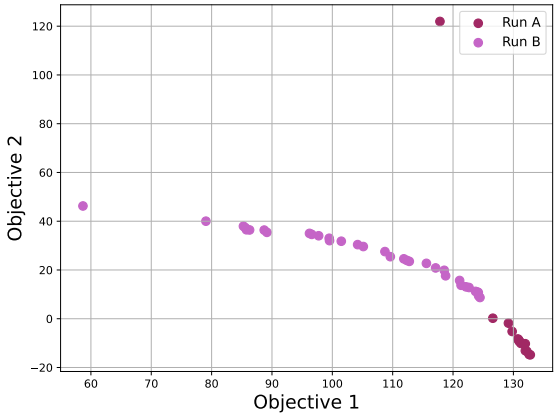


(b) mo-hopper-2d-v4

Figure A.1.: Performance of EMOS compared to state-of-the-art MORL algorithms in the *mo-halfcheetah-v4* domain and in the *mo-hopper-2d-v4* domain. These algorithms were compatible in terms of the environment and they all share a multi-policy approach.



(a) Hypervolume



(b) Pareto fronts

Figure A.2.: Reliability of MORL metrics based on an example of two EMOS runs. A high hypervolume or expected utility does not always guarantee a well-formed Pareto front, and a visually good Pareto front may not always reflect high metric values.

List of Figures

Bibliography

[ABR+23] Lucas N Alegre, Ana LC Bazzan, Diederik M Roijers, Ann Nowé, and Bruno C da Silva. Sample-efficient multi-objective learning via generalized policy improvement prioritization. *arXiv preprint arXiv:2301.07784*, 2023.

[Ach18] Joshua Achiam. Spinning up in deep rl. `https://spinningup.openai.com/en/latest/algorithms/vpg.html`, 2018. OpenAI Spinning Up, Retrieved on 9th of January 2025.

[AFT+22] Lucas N Alegre, Florian Felten, El-Ghazali Talbi, Grégoire Danoy, Ann Nowé, Ana LC Bazzan, and Bruno C da Silva. Mo-gym: A library of multi-objective reinforcement learning environments. In *Proceedings of the 34th Benelux Conference on Artificial Intelligence BNAIC/Benelearn*, volume 2022, page 2, 2022.

[AL21] Hussain Alibrahim and Simone A Ludwig. Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1551–1559. IEEE, 2021.

[DPT22] Daniele Di Pompeo and Michele Tucci. Search budget in multi-objective refactoring optimization: a model-based empirical study. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 406–413. IEEE, 2022.

[ETT12] Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. *The MIT Press*, 2012.

[FAN+24] Florian Felten, Lucas N Alegre, Ann Nowe, Ana Bazzan, El Ghazali Talbi, Grégoire Danoy, and Bruno C da Silva. A toolkit for reliable benchmarking and research in multi-objective reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[Fel24] F. Felten. *Multi-Objective Reinforcement Learning*. Doctoral thesis, Unilu - Université du Luxembourg, 2024.

[GZLP23] Thomas Gabor, Maximilian Zorn, and Claudia Linnhoff-Popien. Lecture computational intelligence definition sheet. *LMU Munich*, 2023.

[HAA+19] Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, and VB Surya Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, 10(12):390, 2019.

[HRB+22] Conor F Hayes, Roxana Rădulescu, Eugenio Bargiacchi, Johan Källström, Matthew Macfarlane, Mathieu Reymond, Timothy Verstraeten, Luisa M Zintgraf, Richard Dazeley, Fredrik Heintz, et al. A practical guide to multi-objective reinforcement learning and planning. *Autonomous Agents and Multi-Agent Systems*, 36(1):26, 2022.

[KT18] Shauharda Khadka and Kagan Tumer. Evolution-guided policy gradient in reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[LHY23] Haoye Lu, Daniel Herman, and Yaoliang Yu. Multi-objective reinforcement learning: Convexity, stationarity and pareto optimality. In *The Eleventh International Conference on Learning Representations*, 2023.

[Li17] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[LPX+24] Ruohong Liu, Yuxin Pan, Linjie Xu, Lei Song, Pengcheng You, Yize Chen, and Jiang Bian. C-morl: Multi-objective reinforcement learning through efficient discovery of pareto front. *arXiv preprint arXiv:2410.02236*, 2024.

[MSG99] David E Moriarty, Alan C Schultz, and John J Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, 1999.

[RV18] Yampolskiy RV. Why we do not evolve software? analysis of evolutionary algorithms. *Evol Bioinform Online*, 2018.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[TKT+24] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024.

[VDB+11] Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning*, 84:51–80, 2011.

[VMN14] Kristof Van Moffaert and Ann Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512, 2014.

[Waw09] Paweł Wawrzyński. A cat-like robot real-time learning to run. In *Adaptive and Natural Computing Algorithms: 9th International Conference, ICANNGA 2009, Kuopio, Finland, April 23-25, 2009, Revised Selected Papers 9*, pages 380–390. Springer, 2009.

[WHT20] Yuhui Wang, Hao He, and Xiaoyang Tan. Truly proximal policy optimization. In *Uncertainty in artificial intelligence*, pages 113–122. PMLR, 2020.

[Wil24] Marcus Williams. Multi-objective reinforcement learning from ai feedback. *arXiv preprint arXiv:2406.07295*, 2024.

[XTM+20] Jie Xu, Yunsheng Tian, Pingchuan Ma, Daniela Rus, Shinjiro Sueda, and Wojciech Matusik. Prediction-guided multi-objective reinforcement learning for continuous robot control. In *International conference on machine learning*, pages 10607–10616. PMLR, 2020.

[ZDA23] Dan Zhou, Jiqing Du, and Sachiyo Arai. Efficient elitist cooperative evolutionary algorithm for multi-objective reinforcement learning. *IEEE Access*, 11:43128–43139, 2023.

[ZKR+15] Luisa M Zintgraf, Timon V Kanters, Diederik M Roijers, Frans Oliehoek, and Philipp Beau. Quality assessment of morl algorithms: A utility-based approach. In *Benelearn 2015: proceedings of the 24th annual machine learning conference of Belgium and the Netherlands*, 2015.