UNIVERSITY OF PRINCE EDWARD ISLAND

Deep Reinforcement Learning For Smart Restarts In Exploration-only Exploitation-only Metaheuristic Hybrids

by

Bowen Xu

A thesis submitted in partial fulfillment for the degree of Master in Computer Science

in the

Faculty of Science

School of Mathematical and Computational Science

August 2024

THESIS/DISSERTATION NON-EXCLUSIVE LICENSE

Family Name: Xu	Given Name, Middle Name (if applicable): Bowen
Full Name of University: University of Prince Edward Island	
Faculty, Department, School: Faculty of Science, School of Mathematical and Computational Sciences (SMCS)	
Degree for which thesis/dissertation was presented:	Date Degree Awarded:
Thesis/dissertation Title: Deep Reinforcement Learning for smart restarts in exploration-only exploitation-only metaheuristics hybrids	
Date of Birth. It is **optional** to supply your date of birth. If you choose to do so please note that the information will be included in the bibliographic record for your thesis/dissertation.	

In consideration of my University making my thesis/dissertation available to interested persons, I, Bowen Xu, hereby grant a non-exclusive, for the full term of copyright protection, license to the University of Prince Edward Island to :

(a) to archive, preserve, produce, reproduce, publish, communicate, convert into any format, and to make available in print or online by telecommunication to the public for non-commercial purposes;
(b) to sub-license to Library and Archives Canada any of the acts mentioned in paragraph (a).

I undertake to submit my thesis/dissertation, through my University, to Library and Archives Canada. Any abstract submitted with the thesis/dissertation will be considered to form part of the thesis/dissertation.

I represent that my thesis/dissertation is my original work, does not infringe any rights of others, including privacy rights, and that I have the right to make the grant conferred by this non-exclusive license.

If third party copyrighted material was included in my thesis/dissertation for which, under the terms of the *Copyright Act*, written permission from the copyright owners is required I have obtained such permission from the copyright owners to do the acts mentioned in paragraph (a) above for the full term of copyright protection

I retain copyright ownership and moral rights in my thesis/dissertation, and may deal with the copyright in my thesis/dissertation, in any way consistent with rights granted by me to my University in this non-exclusive license.

I further promise to inform any person to whom I may hereafter assign or license my copyright in my thesis/dissertation of the rights granted by me to my University in this non-exclusive license.

Signature [redacted]	**Date** Jul. 26th, 2024

University of Prince Edward Island

Faculty of Science

Charlottetown

CERTIFICATION OF THESIS WORK

We, the undersigned, certify that Bowen Xu, candidate for the degree of Master of Science in the School of Mathematical & Computational Sciences at the University of Prince Edward Island, has presented a thesis with the following title: "Deep Reinforcement Learning for Smart Restarts In Exploration-only Exploitation-only Metaheuristic Hybrids". We certify that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate through an oral examination held on July 26th, 2024.

Examiners:

Dr.Antonio Bolufé-Röhler (Supervisor)

Dr. Paul Sheridan (Committee member)

Dr. Christopher Power (Committee member)

Dr. Kuljeet Grewal (External examiner)

Dr. J. Patrick Murphy (Chair)

[redacted]

Date: 7/26/24

Declaration of Authorship

I, Bowen Xu, declare that this thesis titled, ‘Deep Reinforcement Learning For Smart Restarts In Exploration-only Exploitation-only Metaheuristic Hybrids’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Bowen Xu

Date: August 8th, 2024

Abstract

Metaheuristic algorithms excel in addressing challenging optimization problems but often face the issue of premature convergence, limiting their potential during extended optimization periods. This research aims to overcome this limitation by integrating Reinforcement Learning to implement intelligent restart mechanisms in metaheuristic processes. The objective is to enhance the algorithms' ability to explore and exploit the solution space more effectively, thereby improving performance in complex optimization scenarios.

The study starts with a review of current metaheuristic algorithms, highlighting the issue of premature convergence. It then explores Reinforcement Learning principles, particularly their decision-making capabilities, to optimize metaheuristic performance. A novel framework is proposed where Reinforcement Learning agents monitor the optimization process, identify stagnation phases, and initiate intelligent restarts. These restarts are strategically guided by the agents' learned policies, ensuring diversified search when necessary and focused exploration of promising regions.

Experiments on benchmark optimization problems demonstrate that integrating Reinforcement Learning significantly mitigates premature convergence, leading to superior solution quality and robust performance across various domains. This research not only addresses a critical limitation in metaheuristic optimization but also suggests new applications of Reinforcement Learning for enhancing algorithmic efficiency. The findings underscore the potential of intelligent restart mechanisms to transform optimization, enabling more effective and adaptive metaheuristic solutions.

Acknowledgements

I would like to express my deepest appreciation to my supervisor, Dr. Antonio Bolufe-Röhler, for his invaluable guidance, unwavering patience, and insightful feedback throughout the course of this research. His expertise and encouragement have been instrumental in shaping this thesis and my academic growth.

I am also grateful to Dr. Christopher Power, Paul Sheridan, and Kuljeet Grewal for their constructive advice and thoughtful suggestions. Their perspectives have enriched my work, helping me to refine my ideas and approach the subject matter from different angles.

A special thank goes to my parents for their steadfast emotional support over the past three years. Their constant encouragement and belief in my abilities have been a source of strength and motivation, allowing me to persevere through challenges and complete this journey.

Finally, I would like to acknowledge the support of my friends and colleagues, who have provided a source of camaraderie and inspiration. Their understanding and camaraderie have made this academic endeavor an enjoyable and rewarding experience.

Thank you to everyone who has contributed to this journey in ways both big and small. Your support has been invaluable, and I am deeply grateful.

Contents

List of Figures

List of Tables

List of Abbreviations

Acronym	Definition
LP	Linear Programming
IP	Integer Programming
EA	Evolutionary Algorithm
GA	Genetic Algorithm
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
UES	Unbiased Exploratory Search
CMA-ES	Covariance Matrix Adaptation - Evolutionary Strategy
CEC	Congress on Evolutionary Computation
DE	Differential Evolution
PSO	Particle Swarm Optimization
ML	Machine Learning
LS	Local Search
DQN	Deep Q-Network
EEH	Exploration-only Exploitation-only Hybrid
MPS	Minimum Population Search
LSGO	Large Scale Global Optimization
TC	Thresheld Convergence
FE	Function Evaluation
LaF	Leaders and Followers
ReLU	Rectified Linear Unit
StdReward	Standard Reward
NormReward	Normalized Reward

For my mother - Bao Hui Juan

&

For my father – Xu Hua Zhe

Chapter 1

Introduction

Optimization problems are fundamental to numerous applications across various fields, including engineering, economics, and computer science. Traditional optimization algorithms, such as Linear Programming (LP) [1], have been highly successful in solving a wide range of problems. However, these methods often face challenges when dealing with non-differentiable functions, non-convex search spaces with multiple local optima, and large-scale problems with numerous variables or constraints. Such scenarios are common in real-world applications, making the scalability and adaptability of traditional methods a significant concern.

Metaheuristic algorithms, such as Evolutionary Algorithms (EAs) [2], have emerged as powerful tools for addressing these complex optimization problems. Inspired by natural selection processes, these algorithms are adept at performing global searches in complex, multimodal solution spaces. They are particularly valuable in scenarios where the search landscape is irregular, containing multiple optima that traditional methods might struggle to navigate effectively. Despite their strengths, metaheuristics also face limitations, particularly concerning the feasibility and optimality of the solutions they generate. These algorithms, while generally effective at approximating solutions close to the global optimum, do not always guarantee optimality. Moreover, they can suffer from premature convergence, where the search process stagnates at a suboptimal solution, failing to explore the solution space fully [3].

The challenge of premature convergence is a recurring theme in metaheuristic optimization, as it directly impacts the algorithm's ability to find truly optimal solutions. In this thesis, we tackle this issue by exploring strategies that enhance the balance between global exploration and local exploitation within the search process. An overemphasis on local search can lead to the algorithm getting trapped in suboptimal regions, while a focus on global search may slow down convergence. To mitigate these risks, hybridization—integrating various optimization techniques—is often employed. Hybrid algorithms combine the strengths of different methods, such as mathematical programming [4], machine learning [5], and other metaheuristics [6], to create more robust solutions. Many of the most successful real-world and classic optimization solutions have been realized through such hybrid approaches [7, 8].

One of the most promising trends in optimization is the integration of metaheuristics with machine learning. Machine learning models, particularly in supervised learning, have been used to improve the performance of metaheuristics by providing insights into the optimization landscape and helping balance exploration and exploitation more effectively. These hybrid approaches have been successfully applied to a variety of problems, such as the Vehicle Routing Problem [9], Job Scheduling in Cloud Computing [10], and Feature Selection [11]. However, while supervised learning offers several advantages, it also has limitations, particularly in adapting to dynamic or changing problem landscapes [12]. Additionally, supervised learning may not fully leverage the benefits of interacting with the problem space or exploring new solutions.

In response to these challenges, this thesis proposes a novel application of machine learning to improve metaheuristic algorithms through Deep Reinforcement Learning (DRL). Unlike supervised learning, Reinforcement Learning (RL) is well-suited for dynamic environments where the optimal strategy may change over time. In RL, an agent interacts with an environment, learning a policy that maximizes the rewards it receives from performing actions. This ability to learn from interaction makes RL particularly advantageous for overcoming the premature convergence and adaptability issues commonly faced by metaheuristics.

The central hypothesis of this thesis is that metaheuristics can be significantly enhanced by integrating DRL. In this framework, the metaheuristic algorithm acts as the agent, and the objective function represents the environment. The metaheuristic performs actions on the objective function, receives rewards, and learns from these interactions. This learning process allows the algorithm to dynamically adjust its search strategy, potentially overcoming the limitations of traditional metaheuristics, such as premature convergence and suboptimality.

To test this hypothesis, the research focuses on improving a specific metaheuristic: UES-CMA-ES, a combination of Unbiased Exploratory Search (UES) and Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES). UES is designed for effective exploration in multimodal objective functions, while CMA-ES is renowned for its efficiency in optimizing uni-modal functions. The hybridization of these two algorithms leverages the strengths of both exploration and exploitation, making UES-CMA-ES a robust candidate for DRL integration. This explicit separation of exploration and exploitation tasks is particularly conducive to DRL, as it allows the agent to independently and directly control each aspect of the optimization process.

In this thesis, we introduce a novel hybridization approach that combines DRL with UES-CMA-ES, focusing on smart restart strategies. Traditional hybridization methods are set aside in favor of a new version of UES-CMA-ES, designed to incorporate restarts as episodes within the RL framework. This approach enables the agent to learn how to perform restarts more effectively, optimizing the use of function evaluations and improving the overall performance of the algorithm.

The goal of this research is to develop and test a DRL-based hybrid metaheuristic that outperforms traditional methods by avoiding premature convergence and ensuring better exploration of the solution space. This thesis is structured as follows: Chapter 2 provides a comprehensive literature review of metaheuristics and DRL. Chapter 3 presents the formal definition of the multi-restart optimization problem as an RL problem and outlines the initial approach for its solution. In Chapter 4, we discuss the challenges encountered with the initial approach and introduce a redesigned RL environment. Chapter 5 presents a generalized agent capable of solving multiple optimization problems, along with exhaustive results from the IEEE CEC'13 benchmark and an analysis of these results. Finally, Chapter 6 summarizes the research findings, discusses their implications, and outlines future research directions.

Chapter 2

Literature Review

Optimization is a central theme in many scientific and engineering disciplines, and over the past few decades, metaheuristics have become a powerful tool for solving complex optimization problems that are challenging for traditional methods. This chapter provides an overview of metaheuristics, starting with their historical development, key algorithms, and their application to global optimization problems. The discussion will then transition to the emerging trend of hybridizing metaheuristics with machine learning, highlighting the benefits of integrating these techniques. Finally, the chapter will introduce the concept of reinforcement learning and its recent combination with metaheuristics, setting the stage for the novel approach explored in this thesis.

In computer science and mathematical optimization, metaheuristics are considered general-purpose algorithms that can be applied to a wide range of optimization problems [13]. The term “metaheuristics” can be broken down into two components: “meta” refers to an operation at a higher level, controlling other components, while “heuristics” denotes problem-solving techniques designed to find approximate solutions when classical methods are either too slow or fail to find an exact solution. Thus, metaheuristics can be seen as higher-level heuristics that aim to provide reasonably good solutions within a reasonable time frame [14].

One of the key advantages of metaheuristics compared to exact methods is their ability to handle large-scale optimization problems, especially in situations involving incomplete or imperfect information or limited computational resources. While exact methods guarantee optimality, they often struggle with computational complexity, making them impractical for real-world problems. In contrast, metaheuristics are capable of finding satisfactory solutions within a reasonable amount of time, even if they do not guarantee global optimality [13].

The origins of metaheuristics can be traced back to the mid-20th century. In 1945, G. Polya [15] introduced the concept of heuristics in solving optimization problems. Shortly after, in 1947, G. Dantzig [16] developed the simplex algorithm, which can be considered a local search algorithm for linear programming problems. The notion of heuristics gained further prominence in 1971 when J. Edmonds [17] proposed the greedy heuristic in combinatorial optimization literature.

The formal definition of metaheuristics as a specific category of algorithms was introduced by Fred Glover [14] in the 1980s. However, it was during the 1990s that metaheuristics experienced a significant boom in research and applications, with numerous algorithmic variants and adaptations being proposed [18].

Several metaheuristic algorithms have emerged as popular and widely used techniques in the field. These include:

Differential Evolution (DE): DE is a population-based optimization algorithm that utilizes mutation, crossover, and selection operations to explore the search space efficiently [19].

Covariance Matrix Adaptation Evolution Strategy (CMA-ES): CMA-ES is an evolutionary algorithm that adapts a covariance matrix to generate new candidate solutions and effectively search the solution space [20].

Particle Swarm Optimization (PSO): PSO is inspired by the social behavior of bird flocking or fish schooling, where each candidate solution, represented as a particle, adjusts its position based on its own experience and the collective information of other particles [21].

Genetic Algorithm (GA): GA is an evolutionary algorithm inspired by the process of natural selection, where candidate solutions, represented as individuals in a population, evolve over successive generations through operations like selection, crossover, and mutation. This process aims to explore the solution space and optimize a given objective function by retaining the fittest individuals and gradually improving the overall population [22].

These algorithms represent just a fraction of the vast landscape of metaheuristics, and numerous other techniques, such as simulated annealing, genetic algorithms, and ant colony optimization, have been developed and applied in various problem domains [3].

In recent years, the integration of machine learning techniques with metaheuristics has emerged as an important research direction. Hybrid metaheuristics combine algorithmic components from different optimization approaches, often borrowing concepts from other research areas, to improve the efficiency and effectiveness of optimization algorithms [23]. Furthermore, the combination of metaheuristics and machine learning has shown promise in enhancing optimization performance, leveraging the power of data-driven techniques to guide and improve the search process [3].

The field of metaheuristics continues to evolve, driven by ongoing research into algorithmic advancements, hybridization strategies, and innovative applications. With the increasing complexity of real-world optimization problems, metaheuristics provide a valuable toolset for tackling challenges where exact methods fall short.

2.1 Global Optimization

Metaheuristics are applied to a variety of different optimization problems, but in this thesis we will focus on global optimization. Optimization is the selection of a best element, with regard to some criterion, from some set of available alternatives. Optimization problems of sorts arise in all quantitative disciplines from computer science and engineering to operations research and economics, and the development of solution methods has been of interest in mathematics for centuries [24].

Global optimization refers to the process of finding the best possible solution for an optimization problem over the entire feasible solution space. The term “global” in global optimization emphasizes that the objective is to find the global optimum, which is the solution that provides the best objective function value among all feasible solutions. For instance, Travelling Salesman Problem (TSP) is a global optimization problem that objective is to find the shortest possible route that visits a set of cities and returns to the origin city. This is in contrast to local optimization, which focuses on finding the best solution within a limited region of the solution space [25].

Metaheuristic methodologies find wide-ranging applications in addressing diverse optimization problems, and the present thesis places its primary emphasis on the domain of global optimization.

2.1.1 Real World Applications

1. Engineer Design: In the field of engineering, global optimization techniques are used to improve performances, reducing costs and enhancing efficiency. There are many researches of solving engineer design problems by global optimization algorithms, such as improving the shape of micro-aircraft wings design [26], spring design [27] and

welded beam design [28], etc. They are also widely applied in robotics, like producing optimal solutions for robotics systems [29] and multi-robot coordinated exploration [30].

2. Financial Portfolio Optimization: Global optimization is employed to optimize investment portfolios by considering various factors like risk [31], return [32], and diversification [33]. These techniques help in finding an optimal allocation of assets that maximizes returns while minimizing risk.

3. Machine Learning and Neural Network Training: Global optimization is used in training complex machine learning models, such as neural networks [34]. Techniques like Genetic Algorithms [35], Particle Swarm Optimization [36], or simulated annealing help in finding optimal sets of weights and biases that minimize the error function and improve model performance.

4. Image and Signal Processing: Global optimization methods are applied in image and signal processing tasks like image de-noising [37], image segmentation [38], or signal reconstruction [39]. These techniques help in finding the best possible solution among numerous alternatives, considering various constraints and objectives.

5. Drug Discovery and Molecular Modeling: Global optimization algorithms play a crucial role in drug discovery processes, such as molecular docking, virtual screening, or protein structure prediction. These techniques help in searching for the best molecular conformations or drug candidates that optimize desired properties [40].

2.1.2 The IEEE CEC’13 Benchmark

Although there are many real-world applications to global optimization, new algorithms are usually tested first on some well-known benchmarks that allow for a fair comparison between algorithms. In this thesis we will be focusing on the creation of new algorithms and we will use such benchmarks to evaluate our algorithms. Specifically we will use the IEEE CEC’13 Benchmark [41].

The IEEE CEC'13 is a widely recognized benchmark for evaluating the performance of optimization algorithms. It was introduced in 2013 as part of the IEEE Congress on Evolutionary Computation (CEC) competition, which is an annual event where researchers and practitioners showcase their optimization algorithms.

The CEC'13 Benchmark consists of 28 optimization problems, divided into two categories: single-objective and multi-objective optimization. These problems are designed to represent a range of real-world optimization challenges and cover different characteristics such as uni-modal, multi-modal, separable, non-separable, scalable, and non-scalable functions.

2.2 Hybrid Metaheuristics

A hybrid metaheuristic is one that combines a metaheuristic with other optimization approaches, such as algorithms from mathematical programming, constraint programming, and machine learning. Both components of a hybrid metaheuristic may run concurrently and exchange information to guide the search [8].

Over the last years, interest in hybrid metaheuristics has risen considerably in the field of optimization. The best results found for many practical or academic optimization problems are obtained by hybrid algorithms [18]. In this thesis we will focus on the hybridization with ML techniques.

2.2.1 Hybridization with Machine Learning

In the optimization landscape, a substantial amount of data is generated through the search process. The application of ML techniques allows for the efficient management and utilization of this data, substantially refining the optimization procedure. The synergy of machine learning with metaheuristic algorithms has emerged as a focal area of interest,

	No.	Functions	$f_i^*=f_i(x^*)$
Unimodal Functions	1	Sphere Function	-1400
	2	Rotated High Conditioned Elliptic Function	-1300
	3	Rotated Bent Cigar Function	-1200
	4	Rotated Discus Function	-1100
	5	Different Powers Function	-1000
Basic Multimodal Functions	6	Rotated Rosenbrock's Function	-900
	7	Rotated Schaffers F7 Function	-800
	8	Rotated Ackley's Function	-700
	9	Rotated Weierstrass Function	-600
	10	Rotated Griewank's Function	-500
	11	Rastrigin's Function	-400
	12	Rotated Rastrigin's Function	-300
	13	Non-Continuous Rotated Rastrigin's Function	-200
	14	Schwefel's Function	-100
	15	Rotated Schwefel's Function	100
	16	Rotated Katsuura Function	200
	17	Lunacek Bi_Rastrigin Function	300
	18	Rotated Lunacek Bi_Rastrigin Function	400
	19	Expanded Griewank's plus Rosenbrock's Function	500
	20	Expanded Scaffer's F6 Function	600
Composition Functions	21	Composition Function 1 (n=5,Rotated)	700
	22	Composition Function 2 (n=3,Unrotated)	800
	23	Composition Function 3 (n=3,Rotated)	900
	24	Composition Function 4 (n=3,Rotated)	1000
	25	Composition Function 5 (n=3,Rotated)	1100
	26	Composition Function 6 (n=5,Rotated)	1200
	27	Composition Function 7 (n=5,Rotated)	1300
	28	Composition Function 8 (n=5,Rotated)	1400
Search Range: $[-100,100]^D$			

TABLE 2.1: Summary of the 28 CEC'13 Test Functions

particularly due to its ability to incorporate domain-specific knowledge into the optimization strategy, enhancing the search process's effectiveness [42].

The amalgamation of machine learning with hybrid metaheuristics culminates in robust optimization algorithms that capitalize on the strengths of both domains. This integration introduces learning and adaptive capabilities into the metaheuristic framework, significantly improving the efficiency and effectiveness of the optimization process [43].

There are five primary methods of integrating machine learning into metaheuristics: initializing populations intelligently [44], refining fitness evaluation and selection processes [45], aiding in population reproduction [46], enhancing algorithm adaptation [47], and aiding in local search strategies.

Population Initialization: The initial population generation in evolutionary algorithms is traditionally random. This stage sets the stage for exploration, deciding areas to be explored or ignored. An inadequate initial population can lead to clusters that overlook vast regions of the solution space, limiting the algorithm's exploration potential [48]. ML integration at this phase can strategically position the initial population and modify decision vectors for superior solution quality, leveraging historical search data for initial set formation [49, 50]. For instance, ML-assisted bridge optimization has shown promising results in this context [51].

For example, in the context of structural design optimization, historical data from previous design projects, such as the dimensions, material properties, and performance metrics of successful structures, could be used to train a machine learning model. This model can then generate an initial population that is biased towards regions of the search space known to contain high-quality solutions, thereby improving the overall efficiency of the optimization process. Similarly, in the case of machine learning-assisted drug discovery, existing molecular datasets and known effective compounds could serve as training data to guide the initialization of candidate molecules in a genetic algorithm [52].

Fitness Evaluation and Selection: Machine learning can model objective functions or optimize the number of evaluations. It finds widespread application in sports, such as winter sports [53] and athlete selection [54].

Population Reproduction and Variation: This step witnesses ML's role in generating new solutions embedded with problem-specific knowledge. This insight can scale down problem size, assist in solution diversity, convergence, and pinpoint potential solution regions. For instance, S-box optimization leverages this approach in dynamic optimization under uncertainty [55].

Algorithm Adaptation: This involves using ML to refine the entire algorithm's parameters and operators. An example can be found in [55], where machine learning techniques are used to provide an operational advantage when searching the constraint space of hardware designs for cryptographic circuits. In addition, the machine learning component of the proposed hybrids can be trained to detect the need to switch back to exploration and avoid the algorithm from converging, keeping it running in an "infinite mode" for solving dynamic optimization problems under uncertainty conditions. Therefore, machine learning can help metaheuristic algorithms balance exploration and exploitation more effectively and adapt to changing conditions.

Local Search (LS): Machine learning in LS strategizes its design, timing, and control, offering a nuanced approach to various local search methods. An example of this, is the research done by Elgamal et al. [56], where the authors proposed an improved equilibrium optimization algorithm integrating novel local search strategies for feature selection in medical datasets, demonstrating the synergy between metaheuristic optimization and machine learning.

The intersection of machine learning with metaheuristics represents a transformative leap in optimization, offering algorithms that are more adaptable, efficient, and capable of tackling complex, dynamic problems.

2.3 Reinforcement Learning

Reinforcement Learning (RL), distinct from other machine learning paradigms like supervised learning, focuses on how agents can learn optimal strategies through interactions with their environment to maximize cumulative rewards. Unlike supervised learning, which relies on direct feedback, RL uses reinforcement signals as indirect assessments of action quality, necessitating experiential learning by the Reinforcement Learning System (RLS).

In RL, an agent operates within an environment, performing actions that yield rewards. The overarching objective is to develop a policy or strategy that maximizes these rewards, adapting to the environment's dynamics. This process emphasizes the agent's ability to learn from consequences rather than explicit instructions.

2.3.1 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines the strengths of deep learning neural networks with the strategy-oriented focus of reinforcement learning (RL), enabling sophisticated decision-making in complex, uncertain environments. At the core of DRL are deep Q-networks (DQNs), which utilize deep neural networks to estimate the action-value function (Q-function). These networks process the agent's observations and output Q-values for possible actions, guiding the agent toward actions that maximize long-term rewards.

The application of DRL has led to groundbreaking advancements across various domains. In chemistry, DQNs have revolutionized molecule optimization by incorporating chemical domain knowledge [57], surpassing traditional methods. In computer science, a notable achievement of DRL is AlphaGo [58], developed by Google DeepMind. AlphaGo's victory over a professional human Go player in 2016 highlighted DRL's potential in mastering complex strategic tasks.

The fusion of DRL and metaheuristic algorithms has emerged as a dynamic research domain, offering groundbreaking solutions to complex optimization problems. Our research is positioned at the forefront of this evolving field, investigating the incorporation of sophisticated machine learning methods, including deep learning and reinforcement learning, within metaheuristic frameworks. This approach is inspired by pioneering studies like that of Sun et al. [59], which demonstrated the efficacy of semi-supervised learning in surrogate-assisted algorithms, particularly in addressing the complexities of uncertain fitness evaluations and data scarcity. Moreover, our adoption of Reinforcement Learning aligns with a progressive trend in optimization research. This method is anticipated to significantly improve the speed and efficacy of optimization solutions, paving the way for more advanced and efficient computational strategies [60].

2.4 Exploitation-only Exploitation-only Hybrids

Metaheuristic algorithms typically endeavor to balance two key processes: exploration, which involves an extensive search across the solution space to pinpoint promising regions or "attraction basins", and exploitation, focusing on the meticulous refinement of solutions within these identified areas to locate local optima. This dual task, however, presents a challenge, as algorithms proficient in exploration, capable of spanning vast and varied solution landscapes, might lack the nuanced precision required for effective exploitation [61]. Conversely, algorithms adept at exploitation, skilled in intensively probing a localized area, may not possess the broad reach necessary for effective exploration. The inherent trade-off between these two processes often means that an algorithm's strength in one area can be counterbalanced by a limitation in the other, necessitating a strategic approach to optimize their combined effectiveness in solving complex optimization problems [62].

Exploration-only and exploitation-only hybrids address this challenge by dividing the optimization process into two distinct and specialized phases:

Exploration Phase: In this phase, the focus is on broadly scanning the solution space to identify promising regions. This requires an algorithm that can traverse wide areas of the search landscape without getting prematurely trapped in local optima. The objective here is not to find the best solution but to identify areas where good solutions are likely to exist.

Exploitation Phase: Once promising regions are identified, the task shifts to fine-tuning and intensively searching within these regions to locate the local optima. This phase requires an algorithm with strong local search capabilities to effectively hone in on the best solutions within the identified basins of attraction.

The motivation for using exploration-only exploitation-only hybrids is the realization that different algorithms or different configurations of the same algorithm can be more effective at different stages of the search process [63]. By specializing and separating these two phases, the hybrid approach aims to utilize the strengths of different algorithms or strategies, thereby potentially achieving better overall optimization performance than a single algorithm attempting both tasks. This approach is particularly beneficial in dealing with complex problems where the solution landscape is non-trivial, featuring multiple local optima and requiring both a broad search to locate these optima and a deep search to exploit them [63].

2.4.1 Exploration-only algorithms

Exploration plays a crucial role in metaheuristic optimization because it helps to overcome the limitations of local search methods, which can often get trapped in local optima. By exploring the solution space, metaheuristic algorithms aim to discover new and potentially better solutions that may exist outside the immediate neighborhood of the current solutions.

2.4.1.1 Minimum Population Search

Minimum Population Search is a recently developed metaheuristic specifically designed to focus on multi-modal functions and to consider from the beginning the issues that may arise when scaling to LSGO. The key ideas were initially developed for two dimensional problems using only two population members in [64], later generalized for standard dimensions in [65], and scaled towards large scale problems in [66]. For improving performance on multi-modal functions, MPS uses Thresheld Convergence (TC) technique in the sampling methods.

TC is designed to avoid a biased exploration by preventing global and local search steps from occurring simultaneously [67]. In many designs of metaheuristics, large explorative and small exploitative steps are often indistinguishably made during the early (explorative) stage of the search process. TC is designed to solve this problem by controlling the distance (search step) between a parent and its offspring solution [68]. The convergence is "held" back until the last stages of the search process, thus the name *Thresheld*.

The motivation for MPS is to guarantee that new solutions are generated outside the hyperplane formed by the n population members. The design idea is that if new solutions are created merely through the addition of line segments between the population members, then a population smaller than the dimensions of the problem will only produce new solutions inside of a restricted subspace. To avoid this limitation MPS uses a population size equal to the dimensionality of the problem. New solutions are then generated using difference vectors to be inside the $d - 1$ dimensional hyperplane defined by the population; and a full coverage of the search space is achieved by taking a subsequent step that is orthogonal to this hyperplane.

In MPS, the initial solutions are generated by Equation 2.1, assuming that the search space is bounded by the same lower and upper bound in each dimension: s_k is the k^{th}

Algorithm 1 MPS $(\alpha, \gamma, maxFEs)$

```
1:  $P \leftarrow InitialPopulation()$
2:  while $FEs \leq maxFEs$ do
3:      $minStep \leftarrow \alpha \cdot d \cdot \left(\frac{maxFEs - FEs}{maxFEs}\right)^{\gamma}$
4:      $maxStep \leftarrow 2 \cdot minStep$
5:      $x_c \leftarrow Centroid(P)$
6:      for $i = 1 : n$ do
7:          $x_i \leftarrow P(i)$
8:          $f_i \leftarrow UnifRandom(-maxStep, maxStep)$
9:          $fo_i \leftarrow UnifRandom(minOrth, maxOrth)$
10:         $orth_i \leftarrow OrthVector(x_i - x_c)$
11:         $trial_i \leftarrow x_i + f_i \cdot (\frac{x_i - x_c}{\|x_i - x_c\|}) + fo_i \cdot (\frac{orth_i}{\|orth_i\|})$
12:     end for
13:     $P \leftarrow BestSolutions(P, trial)$
14: end while
15: return $\vec{x}_k \in P$ with minimum $y_k$
```

population member, rs_i are random numbers which can be -1 or 1. The threshold is initially set to a fraction of the search space diagonal, and updated over the execution of a metaheuristic by using Equation 2.2. In this equation, min_step_i represents the updated threshold values, $diagonal$ is the main diagonal of the search space, FEs is total number of function evaluations and k is the amount of evaluations used so far. The α and γ are parameters that determines the initial threshold and controls the decay rate, respectively (note: $max\_step = 2 \times min\_step$).

$$s_k = (rs_1 \times bound/2, rs_2 \times bound/2, \ldots, rs_n \times bound/2) \tag{2.1}$$

Then, in each generation a new solution $trial_i$ is created from each population member x_i in two steps. First, each member is used as a parent solution to generate an offspring. The "hyperplane points" are obtained by adding the parent-centroid difference vector to the parent solution. Secondly, the orthogonal step is made by taking a random vector orthogonal $(orth)$ to the parent-centroid difference vector (Eq 2.3). This two-step process for generating the new trail solutions $trial_i$ is represented in Equation 2.3. The scaling factor F_i and O_{step_i} separately determine the direction and size of the difference and the orthogonal vectors.

$$min\_step_i = \alpha \times diagonal \times ([FEs - k]/FEs)^{\gamma} \tag{2.2}$$

$$trial_i = x_i + F_i \times (x_i - x_c) + O_{step\_i} \times orth \tag{2.3}$$

In equation 2.3, x_i is the parent solution and x_c is the centroid of the current population. The F_i factor is drawn with a uniform distribution from $[-max\_step, max\_step]$ ($x_i - x_c$ is normalized before scaling). To ensure that the distance from the new trial solution ($trial_i$) to its parent solution (x_i) stays within the acceptable $[min\_step, max\_step]$ threshold range, the O_{step_i} factor is selected with a uniform distribution from the $[min\_orth_i, max\_orth_i]$ interval. The min_orth_i and max_orth_i values are calculated by Eqs 2.4 and 2.5, respectively. The difference vector $x_i - x_c$ and the $orth$ vector is normalized before scaling. Once the new solutions are created, clamping is performed if necessary, and the best n solutions among the parents and offspring survive into the next generation. A pseudo-code is shown in Algorithm 1.

$$min\_orth_i = \sqrt{max(min\_step_i^2 - F_i^2, 0)} \tag{2.4}$$

$$max\_orth_i = \sqrt{max(max\_step_i^2 - F_i^2, 0)} \tag{2.5}$$

MPS demonstrates remarkable proficiency in exploring solution spaces despite employing a relatively small population size, thereby enhancing the likelihood of convergence even within tight computational budgets. By incorporating Thresheld Convergence, MPS adeptly balances global and local search efforts, enabling the early identification of promising regions and conducting intensive local searches during the final stages if deemed necessary, and optionally employing alternative search strategies. This dynamic fusion of techniques culminates in a highly efficient utilization of available FEs. As a result, Minimum Population Search emerges as an elegant and potent algorithm, adeptly tackling intricate multi-modal problems with simplicity and effectiveness.

2.4.1.2 Leaders and Followers

The success rate of exploration is negatively affected by comparing search solutions and reference solutions which have experienced different amounts of exploitation. In general, an exploratory solution will not have experienced exploitation, so an apples-to-apples, or like-to-like comparison will require the reference solution to be quite similar to an exploratory solution. If the success rate of exploration depends on having reference solutions that are similar to exploratory solutions, then either the reference solutions cannot improve very much over time, or the success rate of exploration will be drastically reduced over time.

A second approach to reduce failed exploration is to avoid the bias towards solutions with high relative fitness (e.g., reference solutions) when they are compared to solutions with low relative fitness (e.g., new sampled solutions). The metaheuristic Leaders and Followers (LaF) [69] was specifically designed with this goal in mind.

The development of Leaders and Followers begins with a detailed study on the Rastrigin function [69]. The leaders are reference solutions which guide the search and the followers are (new) search solutions. The key feature of LaF is the introduction of a new population management scheme which attempts to ensure that all comparisons are performed between solutions which have experienced similar amounts of exploitation. In LaF, the leaders are reference solutions which have experienced exploitation and the followers are search solutions. During a search cycle, the flowers are guided by the leaders, but they are only compared against other followers - this ensures like-to-like comparisons among the search solutions. When the members of the followers population have experienced a similar amount of exploitation as the members of the leaders population, the followers can be measure against the leaders in like-to-like comparisons to determine the new set of leaders that will guide the next search cycle. The pseudo-code for LaF is shown in Algorithm 2.

Algorithm 2 LaF $(popSize, maxFEs)$

```
1:  $L \leftarrow$ Initialize the leaders randomly
2:  $F \leftarrow$ Initialize the followers randomly
3:  while $FEs \leq maxFEs$ do
4:      for $i = 1 : popSize$ do
5:          $leader \leftarrow$ Pick a leader from $L$
6:          $follower \leftarrow$ Pick a follower from $F$
7:          $new \leftarrow createSolution(leader, follower)$
8:          if $f(new) < f(follower)$ then
9:              Substitute $follower$ by $new$ in $F$
10:         end if
11:     end for
12:     if $mean(f(F)) < mean(f(L))$ then
13:         $L \leftarrow mergePopulation(L, F)$
14:         $F \leftarrow$ Reinitialize the followers randomly
15:     end if
16: end while
```

2.4.1.3 Unbiased Exploratory Search

Unbiased Exploratory Search (UES) emerges as a solution to a fundamental challenge in metaheuristic optimization: the inherent bias in exploration. This bias manifests when solutions derived from intensive exploitation efforts become integrated into the algorithm's population. Typically, these exploited solutions are deeply entrenched within their respective attraction basins and may exhibit higher fitness levels compared to exploratory solutions in potentially superior basins. Such a scenario leads to biased comparisons, where the algorithm might erroneously favor less optimal solutions—a phenomenon known as failed exploration. Ideally, a fair comparison occurs when both solutions under consideration are products of exploration, ensuring an unbiased evaluation of their respective merits.

However, in single-population metaheuristics, such as PSO, DE, and MPS, a notable limitation is observed with Thresheld Convergence (TC). In these algorithms, reference solutions tend to gravitate towards local optima over time, even in the absence of explicit exploitation strategies [70]. UES effectively counters this issue by implementing TC for generating new search solutions while simultaneously adopting the two-population approach of LaF. This dual strategy prevents the direct comparison of established reference solutions (leaders) with newly generated ones (followers), thereby maintaining the integrity of the exploration process and mitigating the risk of biased evaluations [71].

UES combines the approaches used in MPS and LaF to avoid failed exploration. The two-population scheme of LaF is used to avoid comparing reference solutions (leaders) against newly sampled solutions (followers). To limit concurrent exploration and exploitation, MPS's sampling method is used instead of LaF's sampling, i.e. a Thresheld Convergence minimum step and an orthogonal step are used to create new solutions [63].

The algorithm starts by randomly initializing the populations of leaders and followers. At each iteration new trial solutions are sampled using information from both populations. The new solutions are compared against the followers and the best ones among both sets are selected as the new population of followers. At the end of each iteration the median fitness of the population of followers is compared against the median fitness of the leaders; a restart is performed if the median of the followers is better. In a restart the two populations are merged and the best solutions become the new leaders; the followers are randomly initialized.

Algorithm 3 UES $(\alpha, \gamma, popSize, maxFEs)$

```
$leaders \leftarrow randomPopulation(popSize)$
$followers \leftarrow randomPopulation(popSize)$
**while** $FEs \leq maxFEs$ **do**
    $minStep \leftarrow \alpha \cdot d \cdot \left(\frac{maxFEs-FEs}{maxFEs}\right)^{\gamma}$
    $maxStep \leftarrow 2 \cdot minStep$
    $x_c \leftarrow centroid(followers)$
    **for** $i = 1 : n$ **do**
        $x_i \leftarrow leaders_i$
        $f_i \leftarrow unifRandom(-maxStep, maxStep)$
        $fo_i \leftarrow unifRandom(minOrth, maxOrth)$
        $orth_i \leftarrow orthVector(x_i - x_c)$
        $trial_i \leftarrow x_i + f_i \cdot \frac{x_i-x_c}{\|x_i-x_c\|} + fo_i \cdot \frac{orth_i}{\|orth_i\|}$
    **end for**
    $followers \leftarrow bestSolutions(followers, trial)$
    **if** $mean(followers) < mean(leaders)$ **then**
        $leaders \leftarrow selectBest(followers, leaders)$
        $followers \leftarrow randomPopulation()$
    **end if**
**end while**
**return** $\vec{x}_k \in leaders \cup followers$ with minimum $y_k$
```

2.4.2 Covariance Matrix Adaptation Evolution Strategy (Exploitation Algorithm)

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a popular metaheuristic algorithm used for numerical optimization problems. It belongs to the class of evolutionary algorithms and is particularly effective for solving continuous, non-linear, and non-convex optimization problems.

The CMA-ES algorithm was developed by Nikolaus Hansen in the late 1990s as an extension of the Evolution Strategy (ES) framework. It combines the concepts of evolutionary computation and estimation of distribution algorithms to efficiently search for the optimal solution in a high-dimensional search space. At its core, CMA-ES maintains a population of candidate solutions, referred to as individuals or solutions. Each solution is represented as a vector in the search space and is evaluated based on a fitness function. The algorithm iteratively updates the population by creating new candidate solutions through the adaptation of a covariance matrix and a mean vector.

The key idea behind CMA-ES is to adapt the distribution of candidate solutions in each iteration, based on the information obtained from the previous iterations. This adaptation is achieved by estimating the covariance matrix of the best-performing solutions and adjusting the mean vector accordingly. By dynamically adjusting the distribution, CMA-ES effectively adapts to the characteristics of the optimization problem, allowing it to explore promising regions of the search space and exploit areas with better solutions. CMA-ES also incorporates a mechanism called "cumulative step-size adaptation" to control the exploration-exploitation trade-off. This mechanism dynamically adjusts the step size to balance between exploring new regions and exploiting the current promising areas in the search space.

Overall, CMA-ES is known for its ability to handle a wide range of optimization problems, including those with noisy or multi-modal fitness landscapes. It has been successfully applied to various fields, such as engineering design, machine learning, and computational biology, where accurate optimization of complex functions is crucial.

2.4.3 The UES-CMA-ES Hybrid

The UES-CMA-ES hybrid algorithm represents an effective approach to tackling complex global optimization problems, combining the strengths of UES and CMA-ES. This hybrid algorithm is particularly relevant in scenarios where optimization tasks involve navigating through intricate search landscapes, often characterized by multiple local optima. Such complexities require a robust strategy that can efficiently explore and exploit the search space to identify the global optimum.

In the initial phase of optimization, UES plays a pivotal role. It merges the capabilities of the Minimum Population Search and Leaders and Followers metaheuristics to enhance the exploration process. While MPS focuses on maintaining a balance between exploration and exploitation through Thresheld Convergence, LaF reduces selection bias in solution comparison by employing a dual-population scheme. This amalgamation in UES leads to a comprehensive and unbiased exploration, effectively scouting out diverse and promising areas in the search space that could harbor optimal solutions.

Once UES has effectively mapped out these promising regions, the hybrid transitions to its second phase, leveraging the CMA-ES for exploitation. CMA-ES is adept at fine-tuning solutions within specific areas, adapting its search parameters based on the distribution of successful solutions. This phase is crucial for intensively refining and converging towards the local optima within the identified promising regions. By integrating the expansive search capabilities of UES with the precision-focused exploitation of CMA-ES, the hybrid algorithm ensures a thorough and effective optimization process, capable of navigating complex optimization landscapes to find the most optimal solutions.

More specifically, the UES-CMA-ES uses a straightforward high-level relay strategy. The hybrid start executing UES with a given percentage of the total budget of function evaluations assigned to it. Afterwards, CMA-ES is started from the best found solution using the remaining function evaluations. In this research we will follow the parameter recommendations from previous EEH [55], thus we will assign a 90% of evaluations to UES and a 10% of evaluations for CMA-ES to perform the final convergence. The Algorithm 4 presents a high level pseudo-code of the UES-CMA-ES hybrid.

Algorithm 4 UES-CMA-ES ($\alpha, \gamma, \sigma, maxFEs$)

```
$leaders \leftarrow UES(\alpha, \gamma, 0.6 * maxFEs)$
**while** $FEs \leq maxFEs$ **do**
    **for** leader in leaders **do**
        $CMA\text{-}ES(leader, \sigma)$
    **end for**
**end while**
**return** best found solution
```

Chapter 3

Multi-Restart Optimization as a Reinforcement Learning Problem - an Initial Approach

In this chapter, we present an initial approach to framing multi-restart optimization as a reinforcement learning problem. This novel perspective aims to enhance the efficiency of metaheuristics by intelligently determining restart points and adapting algorithm parameters. The chapter sets the stage for a comprehensive exploration of this methodology, laying the foundation for subsequent detailed discussions on problem definition and implementation strategies.

3.1 Problem Definition

Metaheuristics with multiple restarts follow a similar high-level structure. First, the metaheuristics is executed and stopped when a certain condition is met; in many cases this condition is the full convergence of the algorithm [72]. Secondly, a perturbation method is used to determine where to restart the search [3]. Optionally, as part of the restart process, the parameters of the algorithm may be changed based on information from the search [73]. Figure 3.1 illustrates the reinforcement learning system that we want to build.

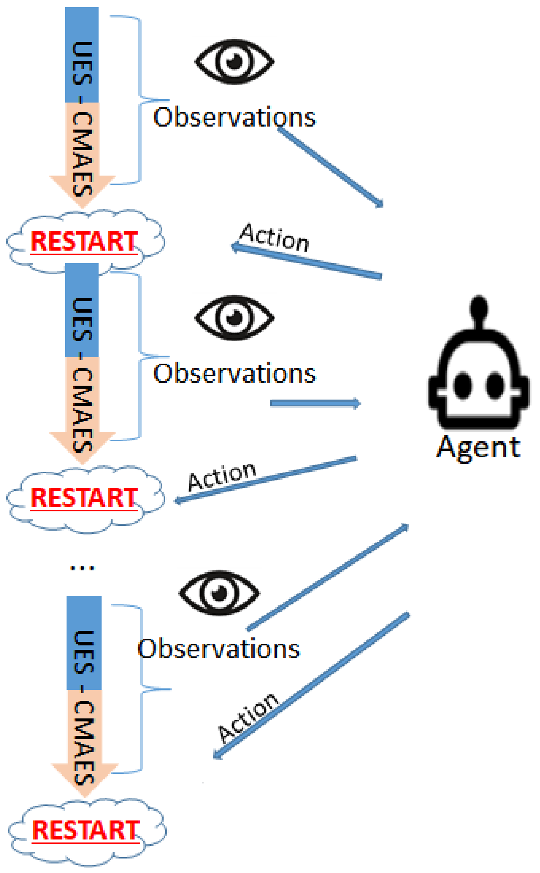

FIGURE 3.1: Illustration of multiple restarts as a reinforcement learning process.

Following this idea, the algorithmic solution to our problem consists of two parts: defining and implementing the deep learning agent and defining and implementing the environment. In this chapter we will introduce an initial approach we used for solving this problem and the limitations of this approach. Chapter 4 will focus on improving this initial solution.

3.2 The Agent

A Deep Q-Network (DQN) agent is a prominent and scientifically significant component within the field of reinforcement learning [74]. It represents a fundamental advancement in the realm of artificial intelligence and autonomous decision-making systems. DQN is a neural network-based algorithm that combines the power of deep learning with reinforcement learning principles to facilitate the training of intelligent agents for sequential decision-making tasks.

The primary objective of a DQN agent is to learn an optimal policy, denoted by a Q-function, which assigns a value to each possible action in a given state. These action values represent the expected cumulative rewards an agent can achieve by taking a specific action from the current state and following an optimal strategy thereafter. DQN agents utilize deep neural networks to approximate the Q-function, allowing them to handle high-dimensional state spaces, such as images or sensor data, effectively.

DQN agents employ a mechanism called experience replay and a target network to stabilize and expedite the learning process. Experience replay involves storing and randomly sampling past experiences to break the temporal correlation of sequential data and enable more efficient learning. Meanwhile, the target network provides a stable target for Q-value estimation during training, mitigating the instability issues commonly encountered in deep reinforcement learning.

In this research, we propose and implement a DQN agent for reinforcement learning tasks using TensorFlow 2. The neural network architecture consists of three densely connected layers with ReLU activation functions, featuring 100, 75, and 50 units, respectively. The output layer employs a linear activation function and is initialized with a Random Uniform initializer. The total number of actions varies, determined by the action specification of the specific environment.

We employ the Adam optimizer with a learning rate of 1×10^{-3} and utilize an element-wise squared loss function for training [75]. The squared loss function, also known as the mean squared error (MSE), is a standard choice in deep learning, particularly in regression tasks. It measures the average of the squares of the differences between the predicted and actual values, which helps in minimizing the prediction error of the neural network. The loss function is crucial for guiding the optimization of the neural network during training, as it quantifies how well the model's predictions align with the target outputs. A lower loss value indicates a better fit of the model to the data [76].

The agent incorporates experience replay using a Reverb replay buffer, enhancing its ability to learn from past interactions. The training loop iteratively collects data using a random policy, trains the Q-network, and evaluates the agent's performance. Results, including training loss, average returns, and best fitness, are recorded and visualized at regular intervals. Our proposed DQN agent demonstrates promising capabilities in learning and decision-making within complex environments. The code for the DQN agent and a complete implementation of all the experiments can be found in the GitHub repository associated to this thesis [77].

3.3 Reinforcement Learning Environment for UES-CMA-ES with multiple restarts

Defining the environment is the most complex part of our solution. It requires the implementation of the UES-CMA-ES hybrid and its integration with a Tensorflow environment. As part of this solution, we also need to define the three main features that define an environment: the observations, the actions and the reward.

3.3.1 Initial Approach - an environment based on selecting the restart position

In our initial approach to this problem we propose a solution that focuses on selecting the restart point without altering the parameters of the metaheuristic. The motivation behind this idea was that we considered the restart point more important than the algorithm's parameters when it comes to restarting a metaheuristics. We were also looking into keeping a simple and straightforward design that could be easily adapted to other metaheuristics.

In order to promote this ease of adaptation we considered using as observations the convergence curve of the metaheuristic (i.e., the fitness of the best found solution in 100 equidistant measurements from beginning to the end). This information can be easily obtained from any optimization algorithm. As actions, the agent would choose one of these 100 solutions as the restart point. The new population for UES will then be randomly generated around this solution. They are generated in an interval that is a fourth of the search range of the search space, using Eq. 3.1, where the $leaders$ are solution generated with a uniform distribution over the entire search space and $start\_point$ is the chosen point for the restart.

$$leaders = leaders/4 + start\_point \tag{3.1}$$

We will refer to this RL solution (environment+agent) as Fit100-Act100 because it uses 100 fitness points as observations and has 100 available actions. More specifically, this initial environment was defined as follows:

- **Observations**: An array with 100 numbers, each number represents the fitness of the best found solution. These 100 measurements are made at equidistant intervals (i.e., every 3,000 function evaluations).
- **Actions**: An integer number between 1 and 100. This number indicates in which solution, from the observations, to restart the algorithm.
- **Rewards**: For the first execution, the reward is made equal to minus the fitness/error of the best found solution. After the first action, the reward represents how much this error was reduced. If the best solution from the new restart is worse

than the previous best found solution then the reward is 0. Eq. 3.2 presents a mathematical definition of the reward function.

$$R = \begin{cases} -\text{error}_{\text{best}} & \text{if it's the first execution} \\ \text{error}_{\text{best}} - \text{error}_{\text{new}} & \text{if the new solution is better} \\ 0 & \text{if the new solution is worse} \end{cases} \tag{3.2}$$

Given this definition of the rewards, the final reward will always be non-positive. This is because the final reward will be equal to minus the error of the best found solution. If the best found solution is the global optimum then the reward will be 0. An advantage of this is that it facilitates the numerical interpretation of the reward, multiplying the reward by -1 will equal the optimization error, which is the most frequently reported value when using the CEC'13 optimization benchmark.

Another advantage is that the agent is not penalized if a given action/restart produces a result that is worse than previously found solutions. This is important because it allows the agent to explore less promising regions of the search space, which in many cases is necessary in order to eventually find better regions [78].

3.4 Experimental Framework for Evaluating an Agent and an Environment

To test the effectiveness of an environment we designed an experimental framework that allow us to evaluate the effectiveness of the Agent + Environment; we will call this combination of an agent plus an environment our Reinforcement Learning solution, or simply RL solution. This experimental framework is the same throughout the thesis for evaluating the effectiveness of an Environment and an Agent. To test the ability of an agent to learn a correct policy we decided to train the agent on individual functions first.

The motivation for this is two fold. First, training over the entire benchmark is very computational expensive, while training on a single function is much faster. Second, learning the right policy from a set of different functions is much more difficult since it requires not only to be able to learn a policy that improves the optimization in a single function but also the ability to generalize and discriminate which policy is the most adequate for each function. Thus, if an agent fails to learn the right policy we wouldn't be able to tell whether it failed because it was unable to learn the right policy for each function or because it was unable to generalize this knowledge for the entire benchmark.

Therefore, the methodology we follow throughout the thesis is to first find an agent plus environment that can effectively learn good policies for individual functions. Once we find this combination of an agent/environment, we then train them on the entire dataset to see how well this RL solution generalizes for a variety of functions.

Even though training on individual functions is less computationally expensive than training over the entire benchmark, the process is still resource-intensive, taking over a week to complete on the Compute Canada cluster. Due to these practical constraints, we selected a subset of six functions that are representative of the complex challenges posed by multi-modal, non-separable, and asymmetrical landscapes. These types of functions are particularly relevant because the EEH hybrids are specifically designed to handle such complexities.

The six benchmark functions selected from the IEEE CEC'13 suite are classic, well-known functions that frequently appear in optimization literature and have been widely used in various studies and benchmarks. These functions were specifically chosen due to their challenging nature, which makes them suitable for assessing the effectiveness of our reinforcement learning solution, particularly in the context of multi-modal optimization. Specifically, the selected functions are:

- **Function 6 'Rotated Rosenbrock's Function' (F6)**: This is a multi-modal non-separable function. The global minimum is inside a long, narrow, parabolic shaped flat valley. To find the valley is trivial. To converge to the global minimum, however, is difficult. State of the art optimization methods are capable of finding the global optimum for this function in 30 dimensions [79].

- **Function 10 'Rotated Weierstrass Function' (F10)**:The Weierstrass function has historically served the role of a pathological function. It was the first published example (1872) specifically created to show that a continuous function may not be differentiable, except on a set of isolated points. This is a multi-modal, asymmetrical, non-separable, function.
- **Function 11 'Rastrigin's Function' (F11)**: The Rastrigin's function is a multi-modal, separable, asymmetrical function wit a very large number of local optima. It is a typical example of non-linear multi-modal function that has been used as the standard benchmark in multiple MPS, LaF and EEH papers [80–82]. Finding the minimum of this function is a fairly difficult problem due to its large search space and its large number of local minima.
- **Function 14 'Schwefel's Function' (F14)**: The Schwefel's function is a highly multi-modal, non-separable and asymmetrical function characterized by having the second better local optimum far from the global optimum. This characteristic has made this function an important benchmark function in a variety of studies [83].
- **Function 16 'Rotated Katsuura Function' (F16)**:The Katsuura function is a multi-modal, non-separable, asymmetrical function that is continuous everywhere yet differentiable nowhere. Despite a high number of very deep local optima, the landscape of this function at a large scale is nearly flat, making it very difficult to optimize [84].
- **Function 19 'Expanded Griewank's plus Rosenbrock's Function' (F19)**: This function is a symmetrical composition of the Griewank's and Rosenbrock functions. Highly multi-modal but globally convex.

Each function was optimized for 30 dimensions - to match many of the results reported in the literature. The total of number of function evaluations used was $FEs = 30 \times 10,000 = 300,000$; this experimental setting corresponds to the evaluation criteria defined for the IEEE CEC'13 benchmark in [41]. The environment was configured to perform 10 restarts, i.e. a restart will be performed every $30,000$ function evaluations.

Two measures were collected in these experiments: the rewards obtained by the agent and the loss of the neural network. To collect information about the reward we pause the training every 500 steps and perform 10 independent executions of the agent, we then report the average value of the rewards. As previously mentioned, this reward is equal to the error from the global optimum (Eq 3.2). To collect the loss we simply report the loss of the agent's neural network at regular steps during the training process. The training is performed for at least $25,000$ steps.

3.5 Computational Results for the Fit100-Act100 reinforcement learning solution

Following the experimental framework described in the previous section we tested the DQN agent using our initial approach, i.e. the environment that chooses the restart position using as observations the fitness of the best found solutions (see Section 3.3.1). In this environment the agent has 100 actions available, each action being one of the 100 solutions' fitness provided in the observation. We trained our RL solution using the experimental framework described in Section 4.3. Figure 3.2 presents the results for functions F6, F10 and F11, while Figure 3.3 presents the results for functions F14, F16 and F19.

As it can be seen in the Figs.3.2 and 3.3 there is no clear improvement in the rewards for any of the functions. Only F6 shows a very slight improvement by the end of the $25,000$ steps. While in F16 there is a more noticeable decrease in the rewards. For the other

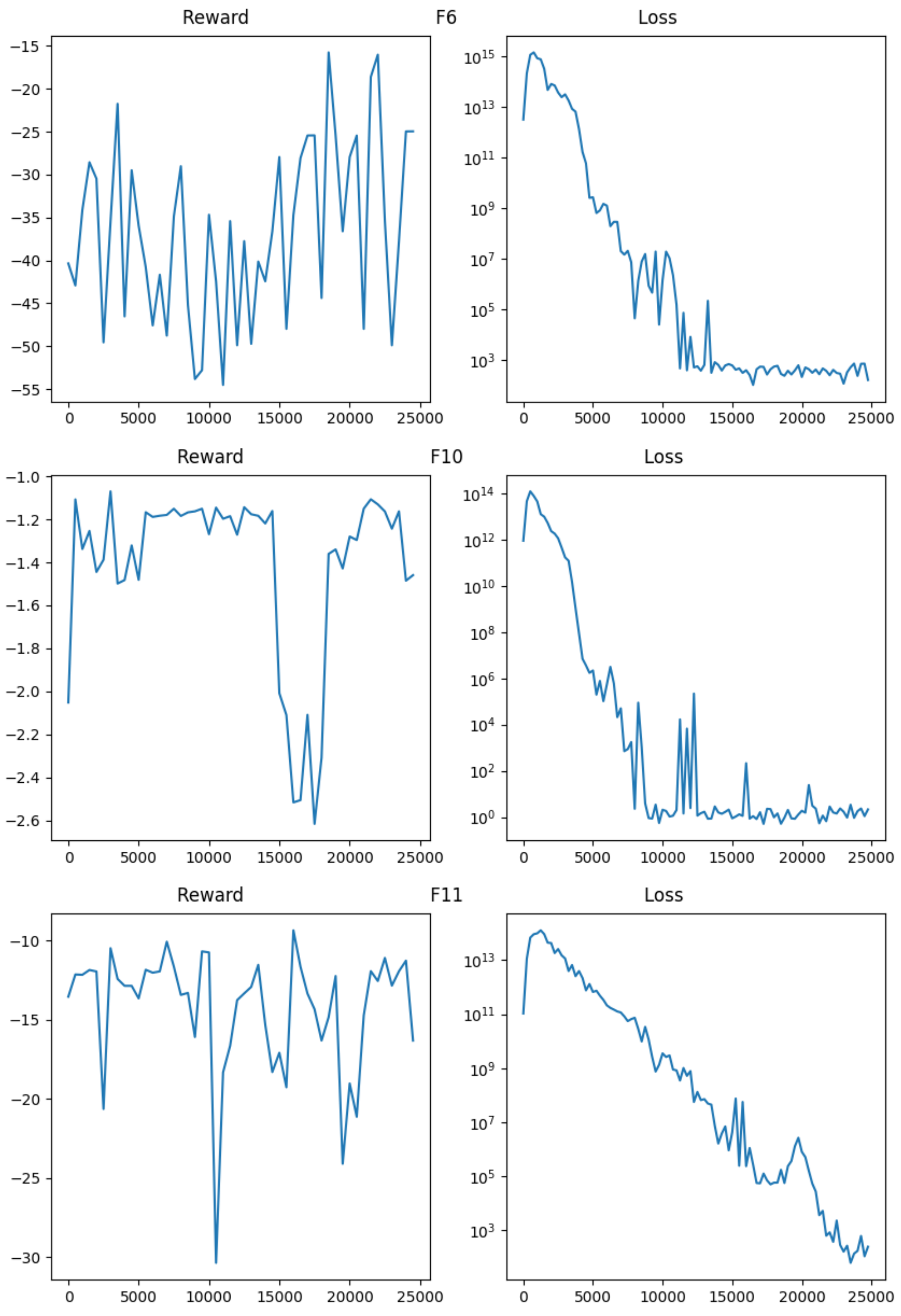

FIGURE 3.2: Rewards and loss for the Fit100-Act100 RL solution when trained on functions F6, F9 and F11.

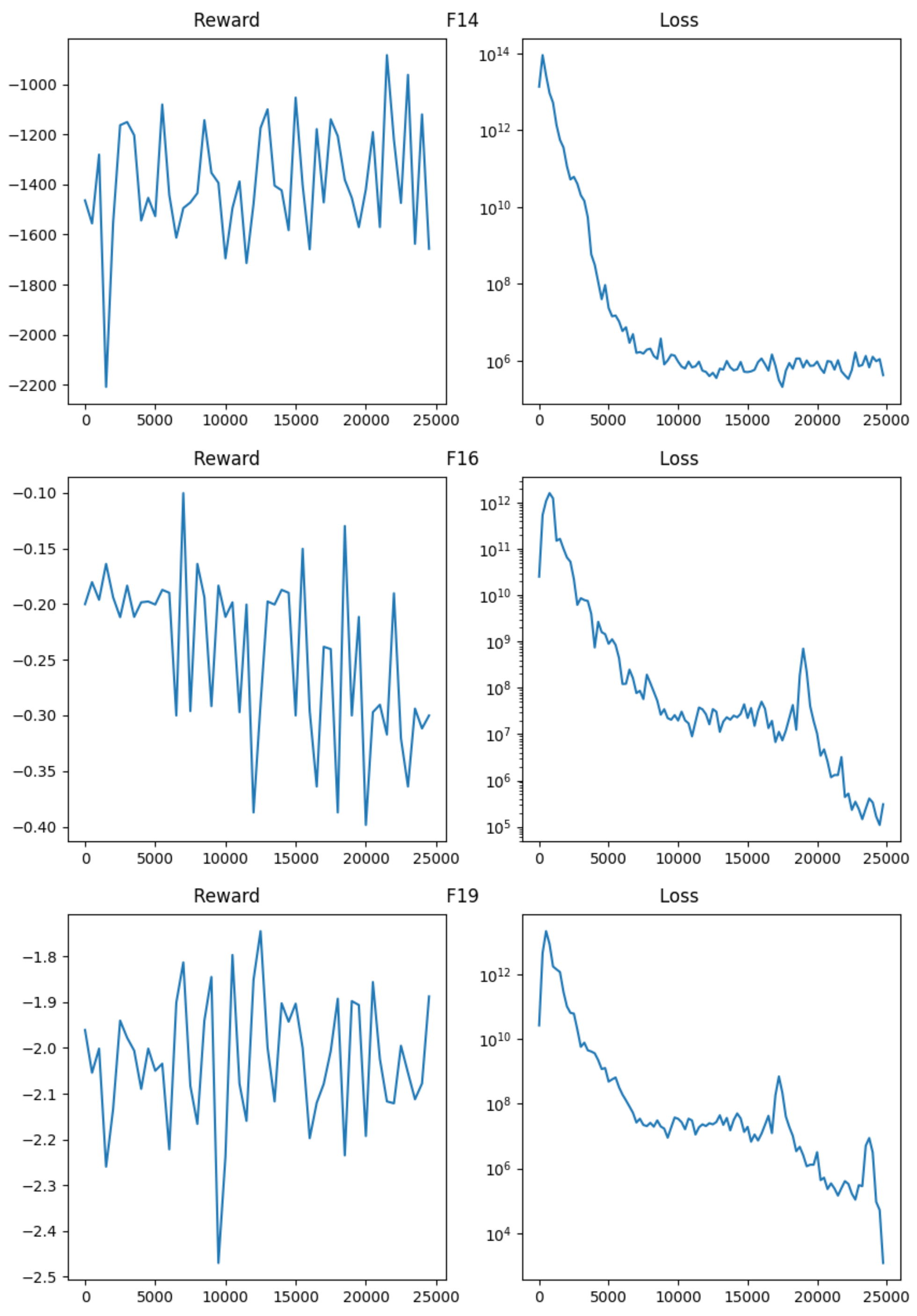

FIGURE 3.3: Rewards and loss for the Fit100-Act100 RL solution when trained on functions F14, F16 and F19.

functions the rewards vary randomly without any clear indication of improvement. However, when looking at the loss function, the DQN model actually reduces the loss as it gets trained.

In deep reinforcement learning, the loss typically refers to the difference between the predicted Q-values and the target Q-values. A lower loss indicates that the model is improving its predictions according to the current policy, but it does not necessarily guarantee better overall performance in reinforcement learning tasks [85]. These results could mean that the policy has settled into a pattern where values can be estimated well by the neural network but for some reason it is not achieving improvements with that policy.

A first reason was that this could be caused by the very large number of actions available to the agent - 100 restart positions to choose from. Thus we decided to test two other RL solutions where we changed the number of available actions to the agent from 100 to 50 and 10; we called this RL solutions Fit100-Act50 and Fit100-Act10, respectively. Notice that the number of observations was not changed, the agent still receives as input the fitness of the best 100 solutions throughout the optimization process. But instead of being able to choose from each of this 100 solutions for restart it only gets to chose from every other solution, in the case of the 50 actions and from every 10 of those solutions in the environment with only 10 actions.

3.6 Computational Results for the Fit100-Act50 and the Fit100-Act10 reinforcement learning solutions

Using the experimental framework from Section 4.3 we repeated the previous experiment for the Fit100-Act50 and the Fit100-Act10 RL solutions. Figs.3.4 and 3.5 show the results for Fit100-Act50 on the selected functions, and Figs.3.6 and 3.7 show the results for Fit100-Act10.

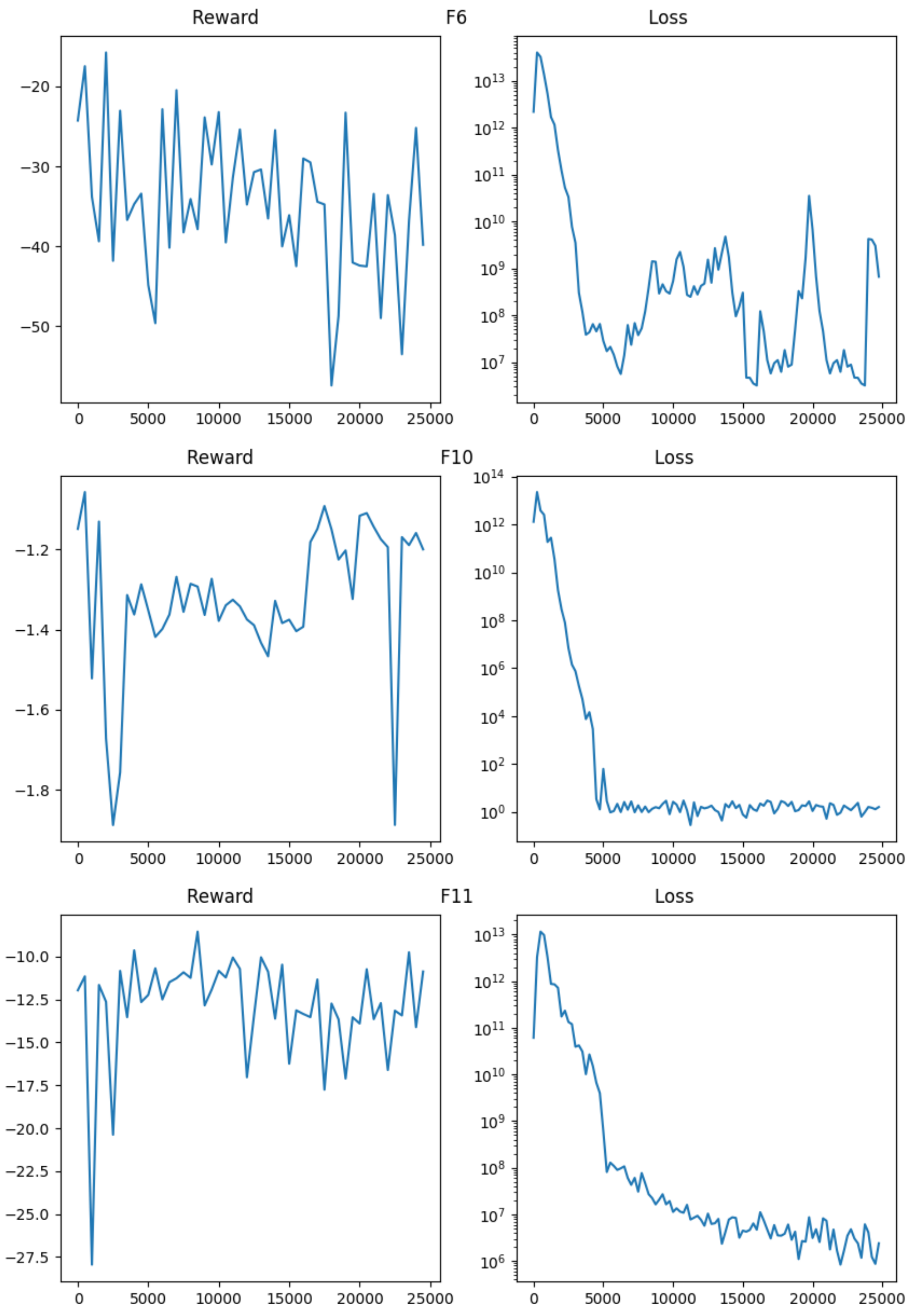

FIGURE 3.4: Rewards and loss for the Fit100-Act50 RL solution when trained on functions F6, F9 and F11.

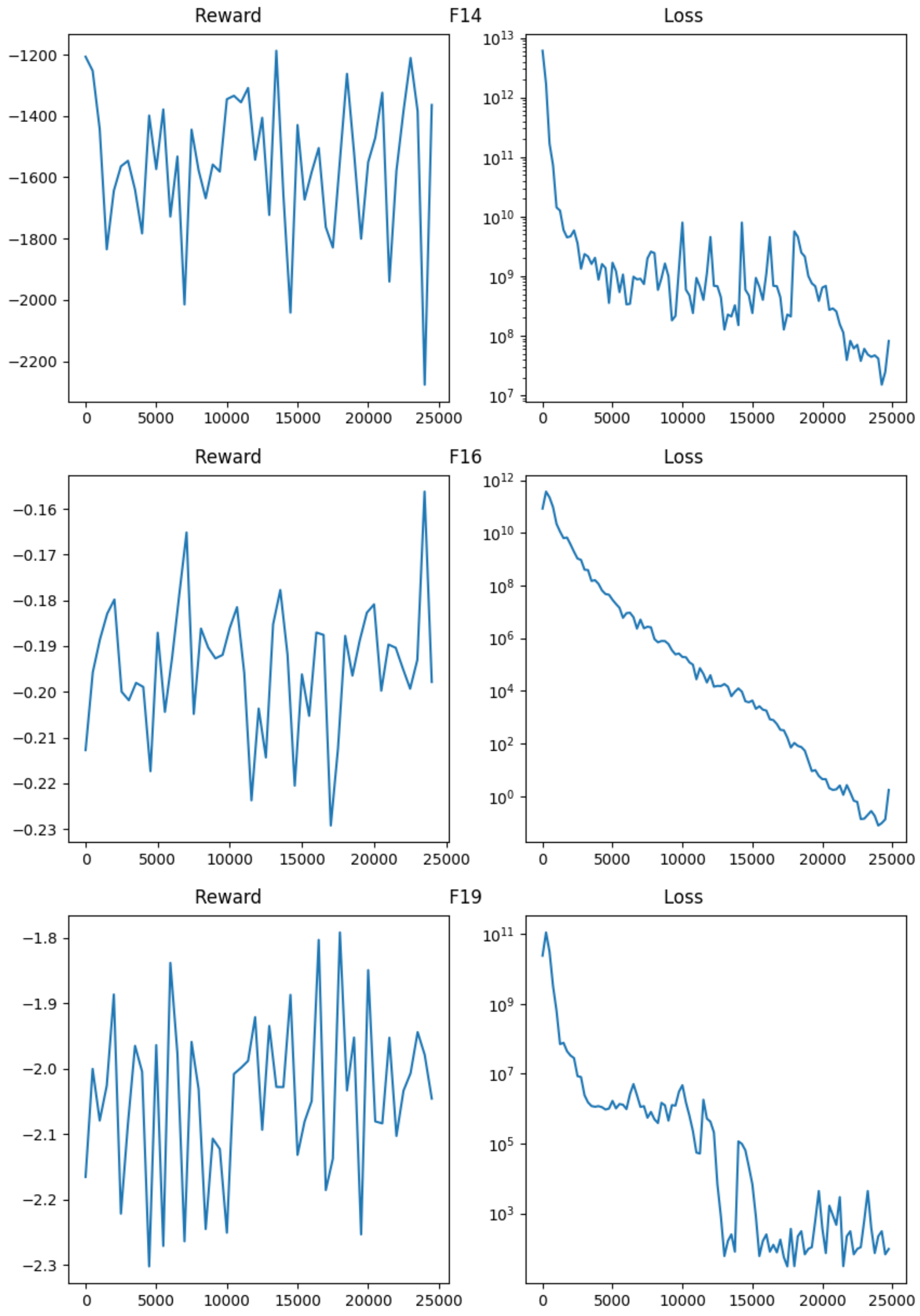

FIGURE 3.5: Rewards and loss for the Fit100-Act50 RL solution when trained on functions F14, F16 and F19.

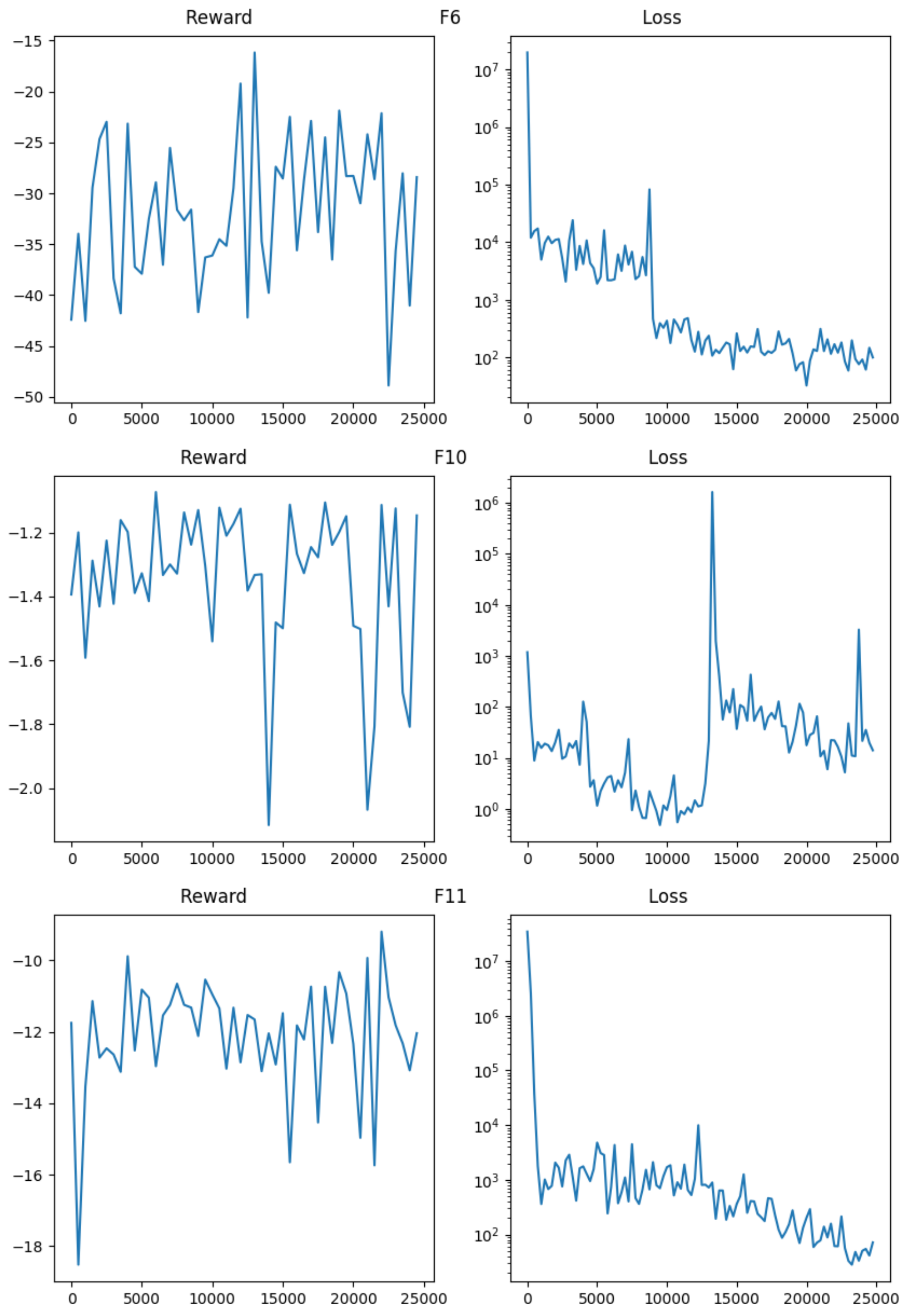

FIGURE 3.6: Rewards and loss for the Fit100-Act10 RL solution when trained on functions F6, F9 and F11.

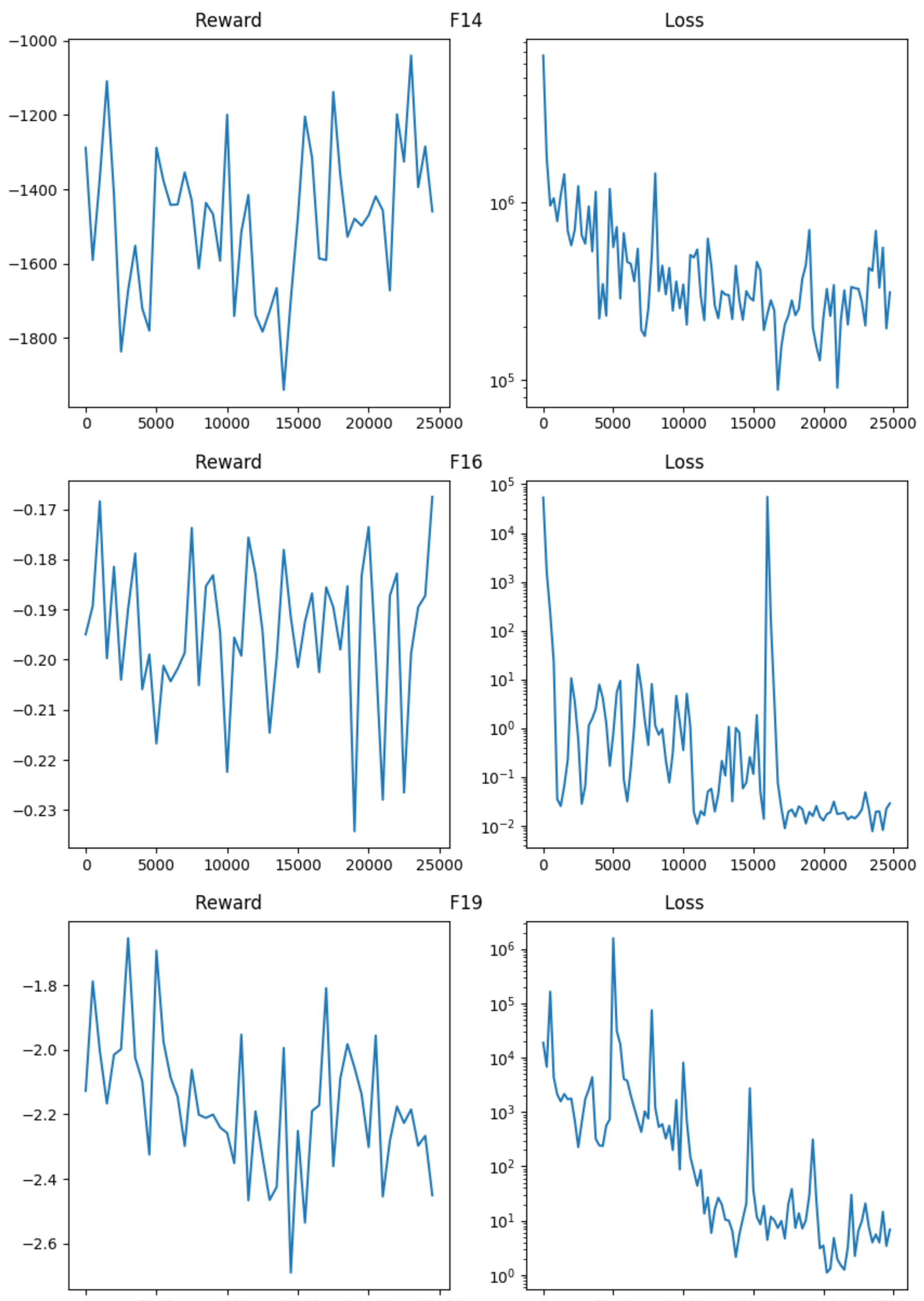

FIGURE 3.7: Rewards and loss for the Fit100-Act10 RL solution when trained on functions F14, F16 and F19.

From these figures it can be noticed that results are very similar to the Fit100-Act100 environment, i.e. we can observe no clear improvement in the rewards achieved by the agent, but the loss is significantly reduced in most functions. This suggest that the number of available actions is not the main cause of the poor performance of this RL solution.

3.7 Discussion

In this chapter we have presented an initial solution to the problem of using reinforcement learning to perform smart restarts for the USE-CMA-ES hybrid. For this initial approach we have defined a DQN agent and environment that takes as input the convergence curve of the optimization algorithm and outputs as action a solution in that curve to be used as the initial point for the restart.

However, this RL solution has not been effective in improving the optimization performance of our hybrid USE-CMA-ES optimizer. The use of a DQN agent is the most common and effective approach when solving complex reinforcement learning problems. Thus, we believe that the problem does not reside in the agent but in the way the environment was designed.

The motivation for this initial environment was two fold. On one hand, we aimed to use as input information that could be easily generalized. That's the reason behind using the convergence curve as observations, since this information is available for any optimization algorithm. However, this information may not be enough for the agent to make an appropriate decision, thus we will explore other alternatives. On the other hand, we assumed that restarting on one of the intermediate solutions of the search process could make the algorithm find better regions in the search space. However, the lack of improvement indicates that this might not be the case. In the next chapter we will focus on designing a set of experiment that will allow us to better understand how the design of the environment affects the restart strategy.

Chapter 4

Multi-restart strategies for UES-CMA-ES - a new approach

In this chapter we analyze the possible reasons behind the failure of the initial approach and we present a new RL solution for the UES-CMA-ES restarts. We will test the environment and policies excluding the agent to find the reason of poor performance of our initial environment. We also created different environments that controls more variables and trained the agent in the most promising ones.

4.1 Analysis of the environment and policies

We designed an experimental framework to analyze the possible causes behind the poor performance of the Fit100-Act100, Fit100-Act50 and Fit100-Act10 RL solutions. The aim of this experiment is to separate the agent from the environment and directly test how a set of selected policies perform on the environment.

By directly testing the policies we can determine how the set of available actions can perform given the environment design and the functioning of the UES-CMA-ES hybrid.

The experiment executes each policy 30 times and plots the reward achieved by each execution of each policy.

4.1.1 Testing the Fit100-Act100 environment

To test the effectiveness of this environment we selected seven policies that represent a variety of search strategies that can be found in the literature [3]. These policies are:

1. **Global search (P1)**: This policy consistently chooses the first reported solution in the search process as the restart point.
2. **Intermediate search (P2)**: This policy consistently chooses a solution in the middle of the search process as the restart point.
3. **Local search (P3)**: This policy consistently chooses the final solution as the restart point. Thus, it focuses more on performing a localized search around the point of convergence.
4. **Uniform incremental convergence (P4)**: This policy starts by choosing the first reported solution, then for the next step chooses the solution reported at the 10% of the search, then 20%, and so on incrementally until the final step chooses the final solution of the search.
5. **Exploration-Exploitation (P5)**: This policy chooses for the first half of the step from the solution at the 33% of the search, for the second half chooses the solution at 66% of the search.
6. **3 Step Exploration-Exploitation (P6)**: This policy chooses the solutions at the 33%, 66% and 99% of the search. Then repeats this 3 times. The final (tenth) step chooses as solution at 99%.
7. **Random Policy (P7)**: This policy randomly chooses any solution for the restart.

Figure 4.1 shows a scatter plot for the performance of 30 independent executions of each policy. The results have been collected for all the functions in the set of basic multi-modal functions in the CEC'13 benchmark, i.e. functions from F6 to F20.

The most noticeable insight from Fig.4.1 is that all the policies perform very similarly. Except for policy 1 (the global search policy), all the other plots show a similar range of results. This result is surprising because the policies represent very different search strategies and their performances should also be expected to differ. It is also noticeable that these selected policies perform very similar to the random policy.

These results indicate a flaw in the design of the Fit100-Act100 environment, this is, that independently of which solution is chosen for the restart, the final results from the optimization algorithm don't vary much. Metaheuristics are stochastic algorithms, thus restarting from the same solution may lead to converging into different areas of the search space, however, the results from Fig.4.1 indicate that this randomness is not sufficient to produce a variety of results.

A more traditional approach used in metaheuristics with multiple restarts is to perform the restart in a neighborhood of the final solution, as it is done in the Iterated Local Search Heuristics [86]. Another idea used in metaheuristics with multiple restarts is changing the parameters of the search after each restart, this is the approach followed by the Variable Neighborhood Search metaheuristic [87]. Using these ideas as inspiration we designed new environments that perform the restart from the final solution.

4.1.2 Testing an environment with variable restart range

We designed a new environment where the actions could control the range of the restart. In this environment the restart is always performed from the final (best) found solution. The new UES solutions for the restart are sampled with uniform distribution around this final solution. Changing the range of the restart means that instead of drawing the

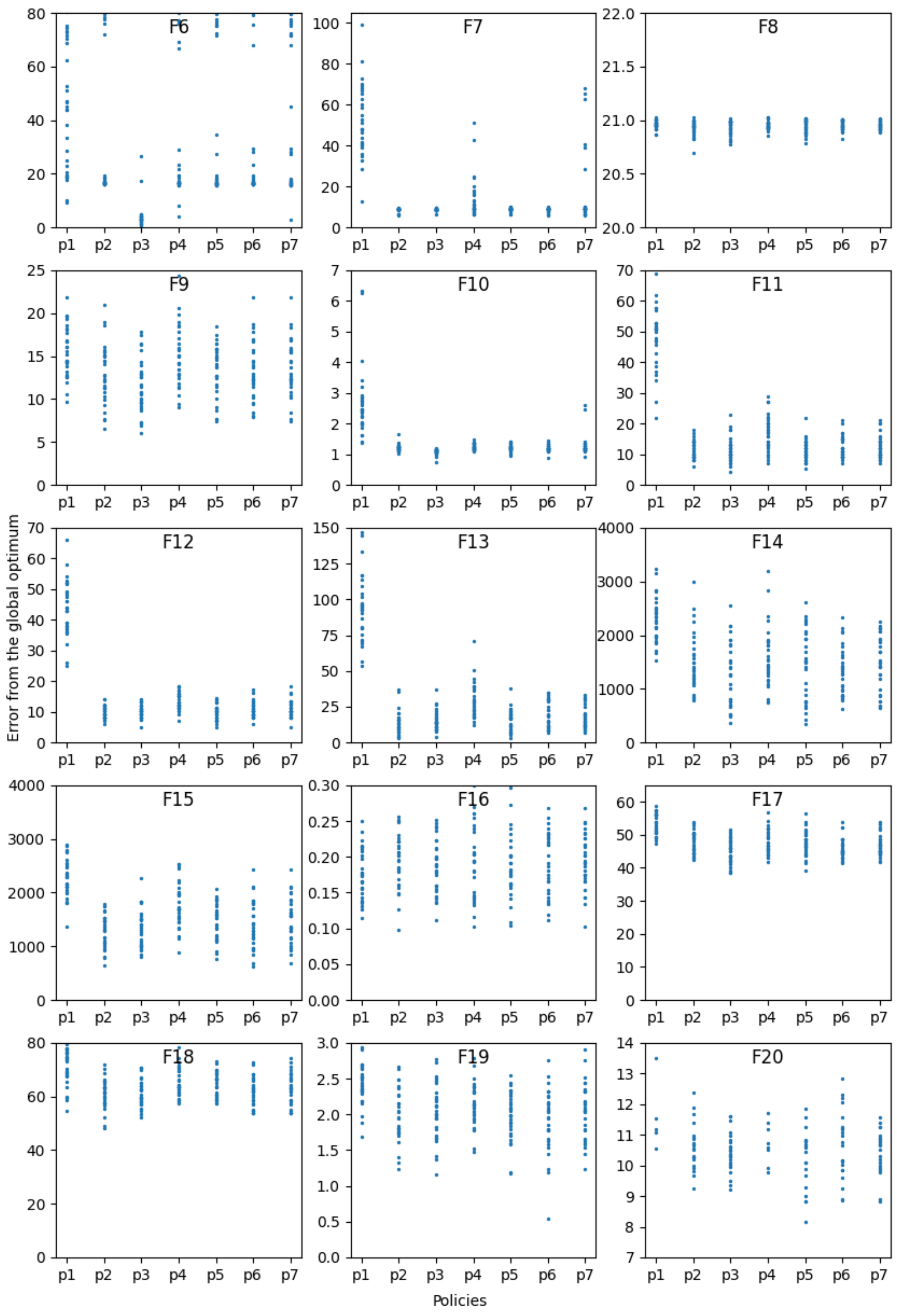

FIGURE 4.1: Test of the Fit100-Act100 environment using selected policies on the basic multi-modal functions.

solutions from a fixed interval range, the agent will be able to control the range on the restart interval.

$$leaders = leaders/(5 \times action + 4) + start\_point \qquad (4.1)$$

Equation 4.1 shows how new solutions are created for the restarts, where the $leaders$ are solution generated with a uniform distribution over the entire search space and $start\_point$ is the chosen point for the restart. The difference with the environment from initial approach is that the range of the initialization is controlled by the $action$ variable, which can take values between 0 and 4. If the action is 0 then the initialization range will be the same as with the initial approach, but as the action value increases the initialization range becomes narrower, leading further restarts to perform a more localized search around the final region of convergence. We will call this test environment *FinalRange*, in reference to the restart occurring always from the final solution and the actions controlling the range of the restart.

To test the effectiveness of this environment we selected seven policies that are similar to those chosen for the Fit100-Act100 environment. The main difference is in the way we interpret the concepts of global and local search. In the Fit100-Act100 environment, global search would refer to selecting a solution from the beginning of the search, i.e. before the optimization process convergences, and local search would refer to restarting in the final/converged solution. In the case of the *FinalRange* environment, global search refers to using a wider restart range (i.e. $action = 4$), while local search refers to performing a more narrow restart (i.e. $action = 0$). The tested policies are:

1. **Global search**: This policy consistently chooses the $action = 0$, performing a restart with a wider range for initializing the new UES solutions.
2. **Intermediate search**: This policy consistently chooses the $action = 2$, performing a restart with an intermediate range for initializing the new UES solutions.

3. **Local search**: This policy consistently chooses the $action = 4$, performing a restart with an narrower/local range for initializing the new UES solutions.

4. **Uniform incremental transition from global to local search**: This policy starts by choosing a global search $action = 0$ for the first two steps and then increasing the action value every two steps until reaching a local search ($action = 4$) at the final two steps.

5. **Alternating Exploration-Exploitation**: This policy chooses alternates between global search ($action = 0$) and local search ($action = 4$).

6. **3 Step Exploration-Exploitation**: This policy chooses actions at 0, 2 and 4 (global, intermediate and local search). Then repeats this 3 times. The final (tenth) step chooses a local search action.

7. **Random Policy**: This policy randomly chooses any action.

Figure 4.2 shows a scatter plot for the performance of 30 independent executions of each policy. The results have been collected for all the functions in the set of basic multi-modal functions in the CEC'13 benchmark, i.e. functions from F6 to F20.

When comparing the results for the *FinalRange* environment vs the *Fit100-Act100* environment, the most noticeable difference is the overall improvement (smaller errors) on most of the functions. Especially noticeable is the improvement in functions F6, F7, F10, F16 and F18, where most policies perform better. In other functions, such as F11, F12, F13, F15 and F20, there is at least one policy that performs much better than all the policies from the *Fit100-Act100* environment, in most cases, this better policy is the Global Search policy (p1).

An important result is that we can see a more noticeable difference between the policies' performance. This is very important because this means that the agent can actually learn which policies perform better. For example, for functions F7, F12 and F15, the global

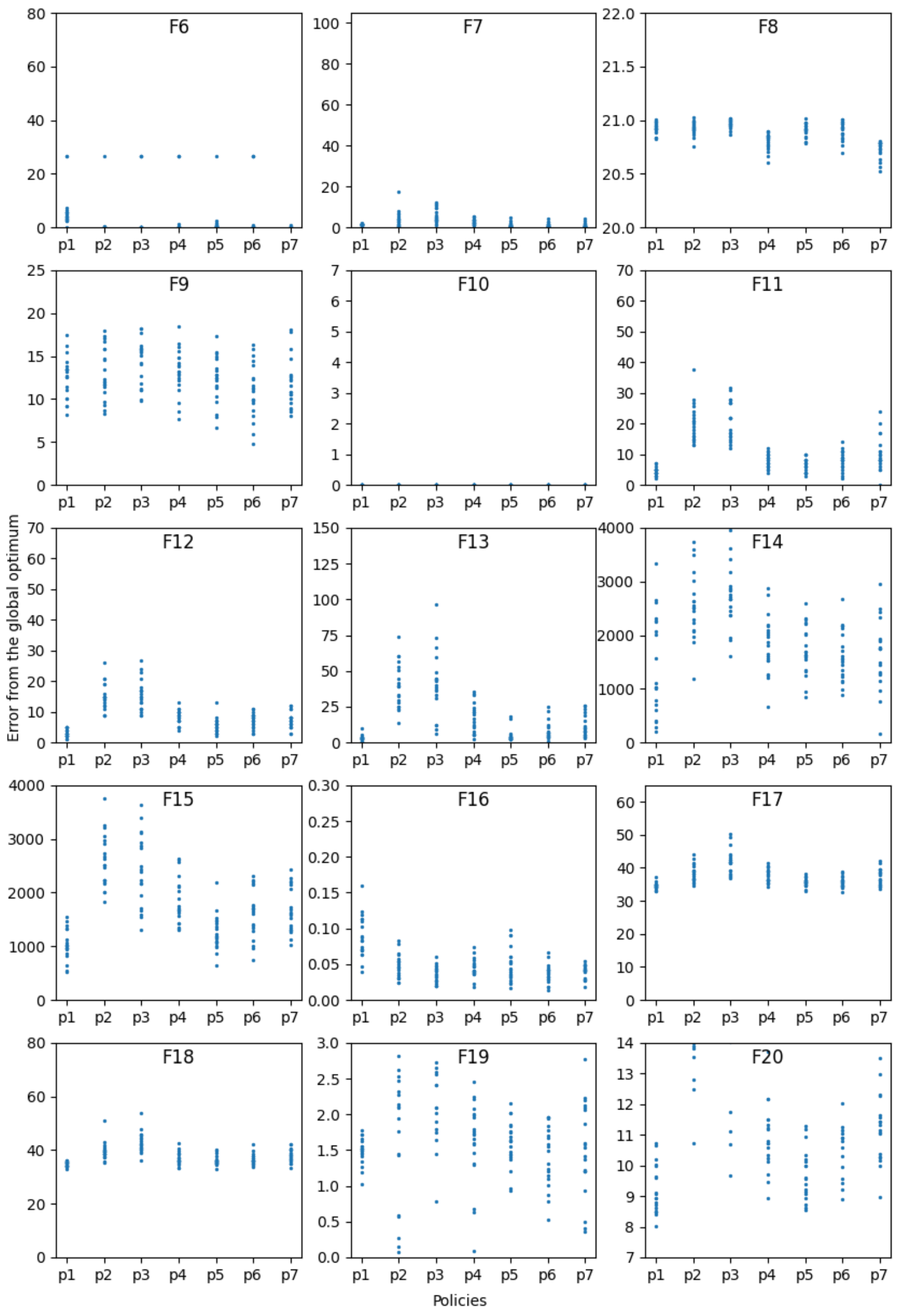

FIGURE 4.2: Test of the FinalRange environment using selected policies on the basic multi-modal functions.

search policy (p1) performs better than the rest, in F16 the best policy is clearly $\sim$ p3. There are however some challenging functions such as F9, F17 and F18, where there is no discernible policy that performs better.

4.1.3 Testing an environment with variable converge rate

A key design decision of UES is the convergence rate, which is controlled by the gamma (γ) parameter in the minimum step equation (see Section 2.4.1.3). Previous research have shown how controlling this parameter during the execution may lead to improved performance for metaheuristics that use a minimum step [88].

With this motivation, we designed an environment where different policies control the convergence rate (γ) on each restart. Following the results presented in [88], we allowed for 5 different actions, each action representing a integer value for γ ranging from 0 to 5. A $\gamma = 0$ means that UES will keep a steady search range without converging, while a $\gamma = 4$ means a fast convergence. As in the previous environment, the restart is performed from the final solution. We have denoted this environment as *FinalRate*, because the restart happens from the final solution and because the convergence rate of UES is the parameter controlled by the policies. The selected policies are:

1. **Global search**: This policy consistently chooses the $action = \gamma = 0$, this means that UES does not converge but performs global search only.
2. **Regular convergence**: This policy consistently chooses the $action = \gamma = 2$, performing a restart with an regular convergence rate ($\gamma = 2$ is the recommended parameter value for UES).
3. **Local search (quick convergence)**: This policy consistently chooses the $action = \gamma = 4$, this makes UES converge quickly after each restart.

4. **Uniform incremental transition from global to local search**: This policy starts by choosing a global search $action = \gamma = 0$ for the first two steps and then increasing the action value every two steps until reaching a local search ($action = \gamma = 4$) at the final two steps.

5. **Alternating Exploration-Exploitation**: This policy chooses alternates between global search ($action = \gamma = 0$) and local search ($action = \gamma = 0$).

6. **3 Step Exploration-Exploitation**: This policy chooses actions at 0, 2 and 4 (global, intermediate and local search). Then repeats this 3 times. The final (tenth) step chooses a local search action.

7. **Random Policy**: This policy randomly chooses any action.

Figure 4.3 shows a scatter plot for the performance of 30 independent executions of each policy. As in the previous experiments, the results have been collected for all the functions in the set of basic multi-modal functions in the CEC'13 benchmark, i.e. functions from F6 to F20.

The results from the *FinalRate* environment are better than the results from the *Fit100-Act100*. When compared to the *FinalRange* we see mixed results: for most functions the *FinalRange* has at least one policy that performs as good or better than the policies from *FinalRate*. However, for functions F8, F17 and F18, the policies $p2$, $p5$ and $p6$, respectively, achieve the best results with the *FinalRate* environment.

In metaheuristic optimization it is common that different parameters settings lead to different results across a variety of functions. Thus, it is not surprising that for some functions the best results are achieved by controlling the convergence rate, even though controlling the range of the restart is in general better. Taking this into consideration we decided to test an environment that can control more than one parameter of the UES-CMA-ES hybrid.

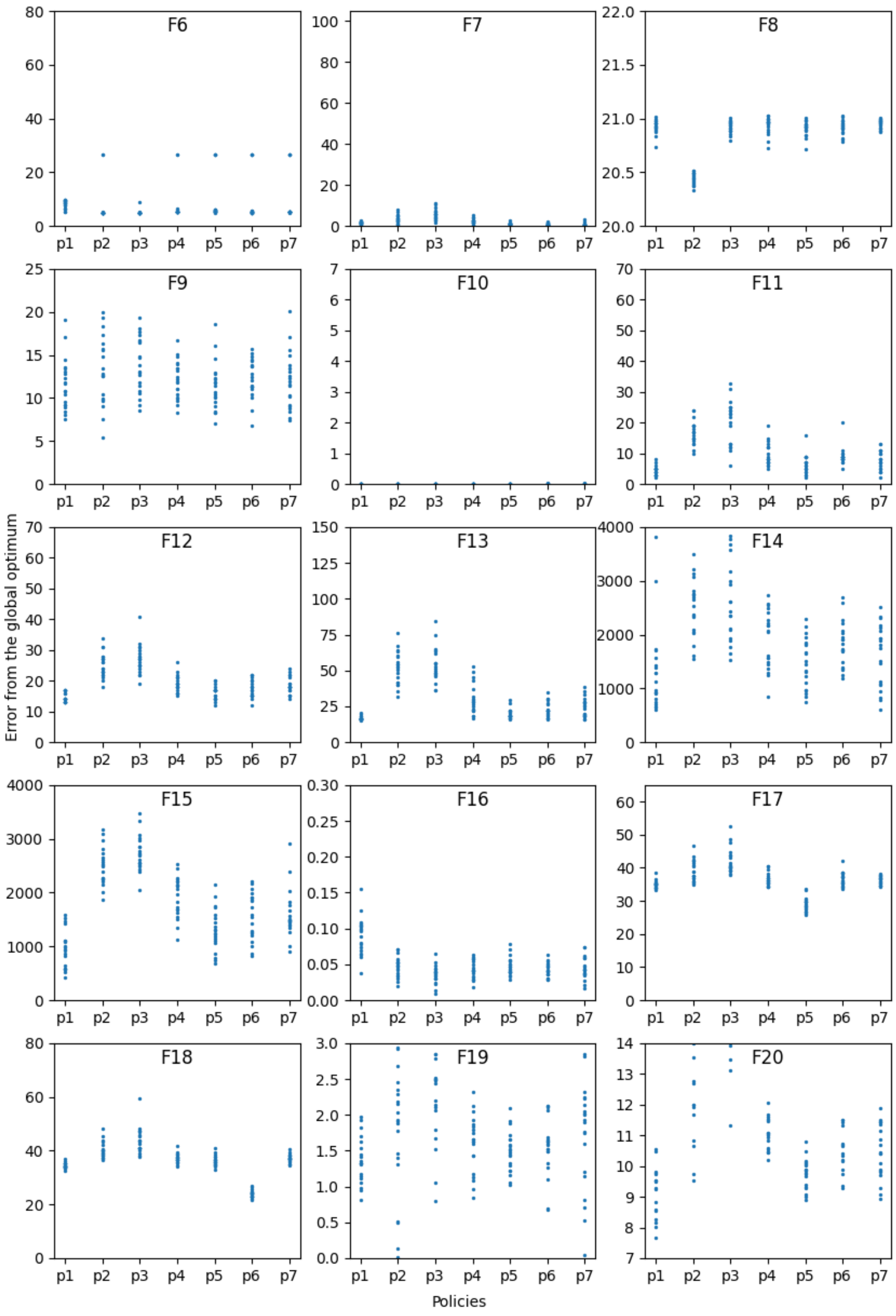

FIGURE 4.3: Test of the FinalRate environment using selected policies on the basic multi-modal functions.

4.1.4 Controlling multiple parameters simultaneously

We designed another environment where the agent can choose between several combinations of parameters that control UES and CMA-ES. The aim with this environment is to take advantage of the benefits that arise from tuning more than one parameter. In this environment the agent has several combinations of parameters available and can choose among these combinations, learning which combination is better suited for a given function. We have named this environment as the *FinalCombo* environment, because restart occur from the final solutions and each action represents changes to a combination of parameters.

The chosen parameters are:

- **Function Evaluations per Algorithm (FEs)**: this parameter determines what percentage of the total amount of function evaluations is assigned to each algorithm. As mentioned in Section 2.4.3 the default distribution used in the previous environment was 90% UES, 10% CMA-ES.
- **Range**: the range parameter controls the range of the restart, it is the same parameter that was controlled in the *FinalRange* environment.
- **Convergence rate** (γ): the convergence rate parameter controls the γ parameter in UES, it is the same parameter that was controlled in the *FinalRate* environment.
- **Alpha** (α): in MPS and UES the α parameter determines the size of the initial threshold (minimum step) of the search (see Section 2.4.1.1).
- **Number of Iterations for UES (iters)**: The iters parameter determines how many iterations (generations) UES performs with its given budget of function evaluations. More iterations allow UES to update the population more times, but comes at the cost of a smaller population.

- **Sigma (σ)**: The σ parameter in CMA-ES determines the initial standard deviation, smaller values mean more local search while larger values lead to a more global search.
- **Population size of CMA-ES (cma_pop)**: This parameter directly controls CMA-ES population size. A larger population size usually leads to more exploration, while a smaller population allows for more generations and a more local/fine search.

Using these parameters we defined twelve different actions, however, only five of those actions were used in the selected policies. For clarity, we only present here those five actions:

- **action 1**: $FEs = 0.9$, $range = 4$, $\gamma = 1$, $\alpha = 0.2$, $iters = 50$, $\sigma = 10$, $cma\_pop = 45$
- **action 2**: $FEs = 0.9$, $range = 0$, $\gamma = 2$, $\alpha = 0.1$, $iters = 40$, $\sigma = 2$, $cma\_pop = 30$
- **action 5**: $FEs = 0.5$, $range = 3$, $\gamma = 3$, $\alpha = 0.1$, $iters = 30$, $\sigma = 1$, $cma\_pop = 15$
- **action 9**: $FEs = 0.5$, $range = 2$, $\gamma = 3$, $\alpha = 0.05$, $iters = 30$, $\sigma = 0.1$, $cma\_pop = 15$
- **action 10**: $FEs = 0.5$, $range = 0$, $\gamma = 4$, $\alpha = 0.01$, $iters = 30$, $\sigma = 0.1$, $cma\_pop = 15$

The first action is a parameter configuration that promotes a strong exploration of UES and CMA-ES, convergence only happens at the very end of CMA-ES execution. Action 2 is also a configuration to promote exploration, but allows UES to converge a bit. Action $\sim$ 5 is a more balanced configuration between global and local search. While actions 9 and

10 are more focused on local search, with action 10 being almost exclusively a local search strategy.

Using these actions we defined a set of policies that are similar to the policies used in the previous environments. Even though this are not necessarily the best policies, we decided to use them in the environment test for a fair comparison of performance. The selected policies are:

1. **Global search**: This policy consistently chooses the $action = [1]$.
2. **Regular convergence**: This policy chooses the following $actions = [1, 5, 5, 5, 5, 5, 5, 10]$
3. **Local search (quick convergence)**: This policy consistently chooses the $action = [10]$.
4. **Uniform incremental transition from global to local search**: This policy chooses the following $actions = [1, 1, 2, 2, 5, 5, 9, 9, 10, 10]$.
5. **Alternating Exploration-Exploitation**: This policy chooses the following $actions = [1, 1, 2, 2, 5, 5, 9, 9, 10, 10]$.
6. **3 Step Exploration-Exploitation**: This policy chooses the following $actions = [1, 5, 10, 1, 5, 10, 1, 5, 10, 10]$.
7. **Random Policy**: This policy randomly chooses any action among the twelve available actions.

Figure 4.4 shows a scatter plot for the performance of 30 independent executions of each policy. As in the previous experiments, the results have been collected for all the functions in the set of basic multi-modal functions in the CEC'13 benchmark, i.e. functions from F6 to F20.

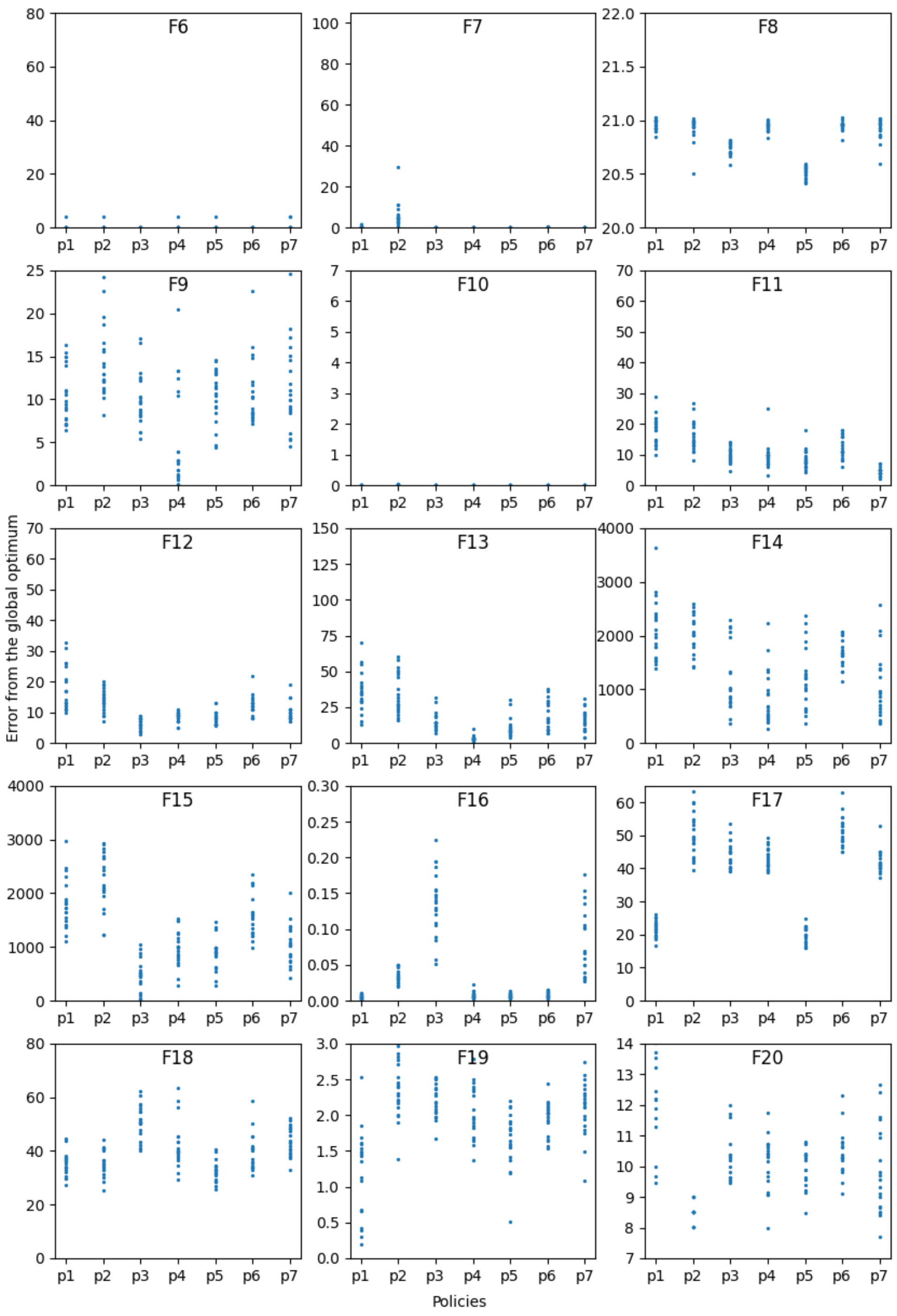

FIGURE 4.4: Test of the FinalRate environment using selected policies on the basic multi-modal functions.

The results for the *FinalCombo* environment are very promising, for all the functions, the results are similar or better than for the two previous environments (*FinalRange* and *FinalRate*). For some functions such as F6, F7, F8, F9, F13, F15, F16, F17 and F20, the results achieved by the best performing policy are remarkably better than the results from the other environment we have tested. Moreover, the diverse performance levels across the chosen policies highlight the potential of crafting a range of distinct search strategies by effectively combining the available actions into various policy configurations. This variability in the performance of the different policies is a very promising feature that may allow the agent to learn better.

4.1.5 Discussion from Testing the Environments

The environment test experiment was designed to analyze the performance of some selected policies. The motivation was to understand the lack of learning of the DQN agent in the Fit100 environment. We were able to determine that this inability to learn was caused by all the policies having similar performance, thus the agent was unable to find suitable policies to improve the performance.

We hypothesized then that this was happening because restarting from an intermediate point in the search using the same search parameters would produce very similar results in every restart. To test this hypothesis we created three new environments, all of which restarted the search from the final solution and changed some parameter(s) of the metaheuristics.

This three environments showed two important improvements over the original Fit100 environment: On one hand, they had more variation in performance among the different policies, this is a promising feature that suggest that an agent could be able to learn the best policies and improve over time. On the other hand, the performance of the selected policies was much better, i.e., the final error was much lower. This is important because it indicates that from an optimization perspective it is more convenient to perform the restart from the final solution and use the agent's actions to choose among the parameters that define the search strategy (as oppose to using the actions to choose the restart point).

From the three new environments, the *FinalRange* and *FinalCombo* show the most promising results. Thus, we chose these two environments for training the DQN agent.

4.2 Training agents on the new environments

In this section we present results for training the DQN agent on the *FinalRange* and *FinalCombo* environments. The experimental framework is the same as presented in Section 4.3 when training on the *Fit100-Act100* environment. We have however, changed the input for the environments.

4.2.1 Observations for the new environments

Metaheuristic algorithms, particularly in the context of optimization, operate in a highly dynamic and often unpredictable search space. The effectiveness of these algorithms is contingent on their ability to adaptively navigate through diverse regions of the search space, balancing the intricate trade-off between exploration and exploitation. The previous set of observations, while effective, might not have fully captured the nuanced variations and subtle shifts in the search landscape experienced during the optimization process. By introducing a new set of observations, we aim to provide the DRL agent with a more comprehensive and complete understanding of the search environment. These new observations are designed to encapsulate a broader spectrum of information, including changes in the population dynamics, convergence rates, and the relative improvements between different phases of the algorithm. This enriched informational context is expected to facilitate a more informed and adaptive decision-making process by the DRL agent, thereby potentially enhancing the overall optimization performance.

It's important to acknowledge that while direct experimental evidence supporting this change is not currently available, the theoretical underpinnings and the logical premise guiding this modification are solid. The new observations are based on well-established principles in both metaheuristics and machine learning, suggesting that they are likely to yield positive results. This change is also reflective of an iterative and evolutionary approach to research, where enhancements are often based on theoretical insights and logical extensions of existing knowledge, followed by empirical validation. Future research, therefore, should aim to empirically investigate the efficacy of these new observations, further refining our understanding and application of DRL in the optimization domain.

The new observation are divided into two types: one type of observations measure the state of the optimization process at regular intervals, providing valuable information about how the search progresses. These observations are made 20 times during the UES optimization, at equidistant steps measured in the number of function evaluations performed. The other type if observation are single measurements made only once during the entire execution of the UES-CMA-ES hybrid.

The regular observations made in 20 states of the optimization are:

- The number of updates to the population of followers, since the previous observation measurement. The number of new solutions added to the population is an indicative of how effective the search is, it has been previously used as an information source for training models for MPS hybrids [89].
- The number of times that the population of followers was restarted, since the previous observation measurement. The number of restart is an important source of information that helps characterize the topology of the objective function. It has been successfully used in previous applications of machine learning for improving the LaF metaheuristics [55].

- The euclidean distance between the best and worse solutions found since the previous observation measurement. This information can be useful to determine the rate of convergence of UES, and thus it could be useful for setting convergence related parameters.

The observations that are only done once per execution of UES-CMA-ES are:

- A Boolean value indicating whether the local search performed by CMA-ES further improved the result obtained by UES.
- The relative improvement (see equation XX) between UES and CMA-ES final solutions.
- The euclidean distance between the restart point and UES final solutions.
- The euclidean distance between the restart point and CMA-ES final solutions.
- The euclidean distance between the CMA-ES initial solution (UES final solution) and CMA-ES final solution.

These 65 observations aim to evaluate whether and how much UES and CMA-ES are moving through the search space and whether this search is yielding good results. This information allows the agent to determine whether the current set of parameters is effective or needs to be adjusted.

4.2.2 Choosing a Set of Optimal Parameter Combinations

The set of parameter combinations (actions of the agent) used in the previous chapter were created to mimic the actions of the policies of the other environment tested. However, this does not mean that those combinations of parameter values were necessarily the best ones. Thus we designed a small experiment intended to test the effectiveness of various parameter combinations.

In this experiment we tested 24 different combination of parameters values. We performed 30 independent executions of the agent for each set of parameter values and used the average of those 30 runs as the evaluation criteria. We tested these combinations on the entire benchmark (i.e., the 28 objective functions). We then counted how many times each combination was among the top 3 for each function. Finally we selected the 12 combinations that most frequently appeared among the top 3. Table 4.1 shows the parameter values of the 12 best combinations.

TABLE 4.1: Best 12 combinations of parameter values

Strategy	**FEs**	**range**	**gamma**	**sigma**	**alpha**	**cma_pop**	**iters**
1	0.9	1	1	0.1	0.1	15	30
2	0.9	0	2	10	0.1	30	30
3	0.9	2	2	0.1	0.05	15	40
4	0.5	3	1	10	0.1	45	50
5	0.5	0	2	10	0.1	15	30
6	0.9	0	3	10	0.1	15	30
7	0.9	0	1	1	0.1	30	40
8	0.9	0	3	0.1	0.1	15	40
9	0.5	4	1	10	0.1	30	30
10	0.5	4	2	1	0.1	15	40
11	0.5	1	1	1	0.1	45	50
12	0.5	2	2	10	0.05	15	30

4.3 Training the DQN agent on the *FinalRange* environment

In this section we present the results for training the DQN agent (see Section 3.2) on the *FinalRange* environment. We have followed the same experimental design as it was used for training the agent in the *Fit100-Act100* environment (see Section 4.3). Figures 4.5 and 4.6 show the plots for the rewards and loss on the functions F6, F9, F11 and F14, F16, F19, respectively.

The most relevant result to notice is that for 4 out of the 6 functions (F6, F11, F14 and F16) a clear improvement in the rewards (optimization performance) is achieved as the agent is trained on the new environment. This result answers the main hypothesis of this

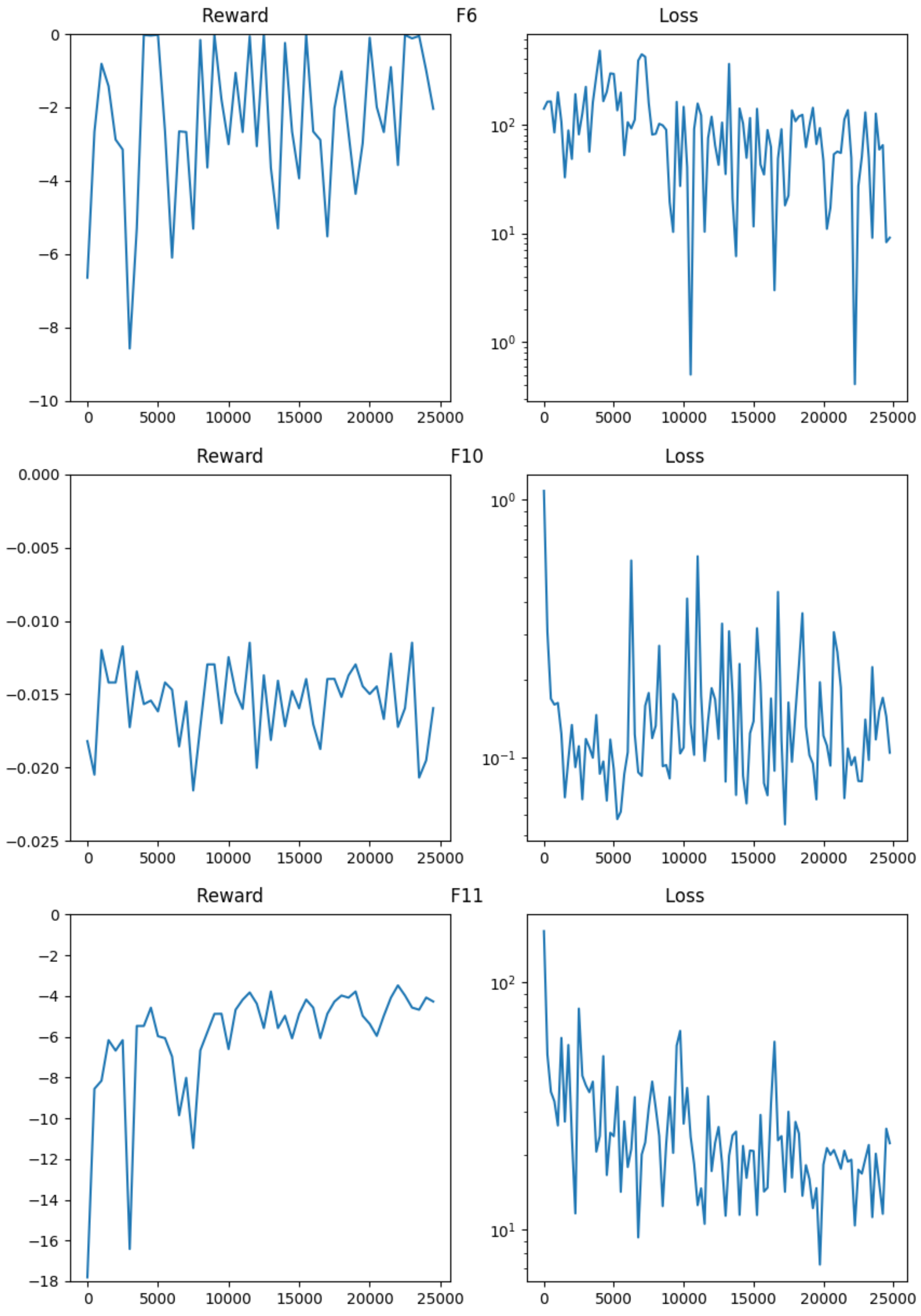

FIGURE 4.5: Rewards and loss for the FinalRange RL solution when trained on functions F6, F9 and F11.

research, of whether a RL agent can be trained to successfully improve the optimization of a multi-start UES-CMA-ES hybrid.

Interestingly, the loss values don't decrease as much as they did for the *Fit100-Act100* environment. This is probably because the new environment, even though more effective from the optimization perspective, is more challenging for the machine learning model. However, the decrease of the loss function observed in all functions is an indication that the agent does learn better policies as the training progresses.

4.4 Training the DQN agent on the *FinalCombo* environment

Following the same experimental framework and the same scale in the figure plots, in this section we present the results for training the DQN agent on the *FinalCombo* environment. The most noticeable result is that the agent achieves a clear improvement in performance for the six functions tested in this experiment. It can also be noticed the clear decrease of the loss function which indicates that the agent's network properly learns the best set of weights for the problem at hand.

By taking a closer look at the reward values in the plots, and comparing them to the rewards from the *FinalRange* environment, we can clearly notice that the results for the *FinalCombo* environment are better. Thus, we will use this reinforcement learning solution, i.e. the DQN agent and the *FinalCombo* environment, for the next step of our research.

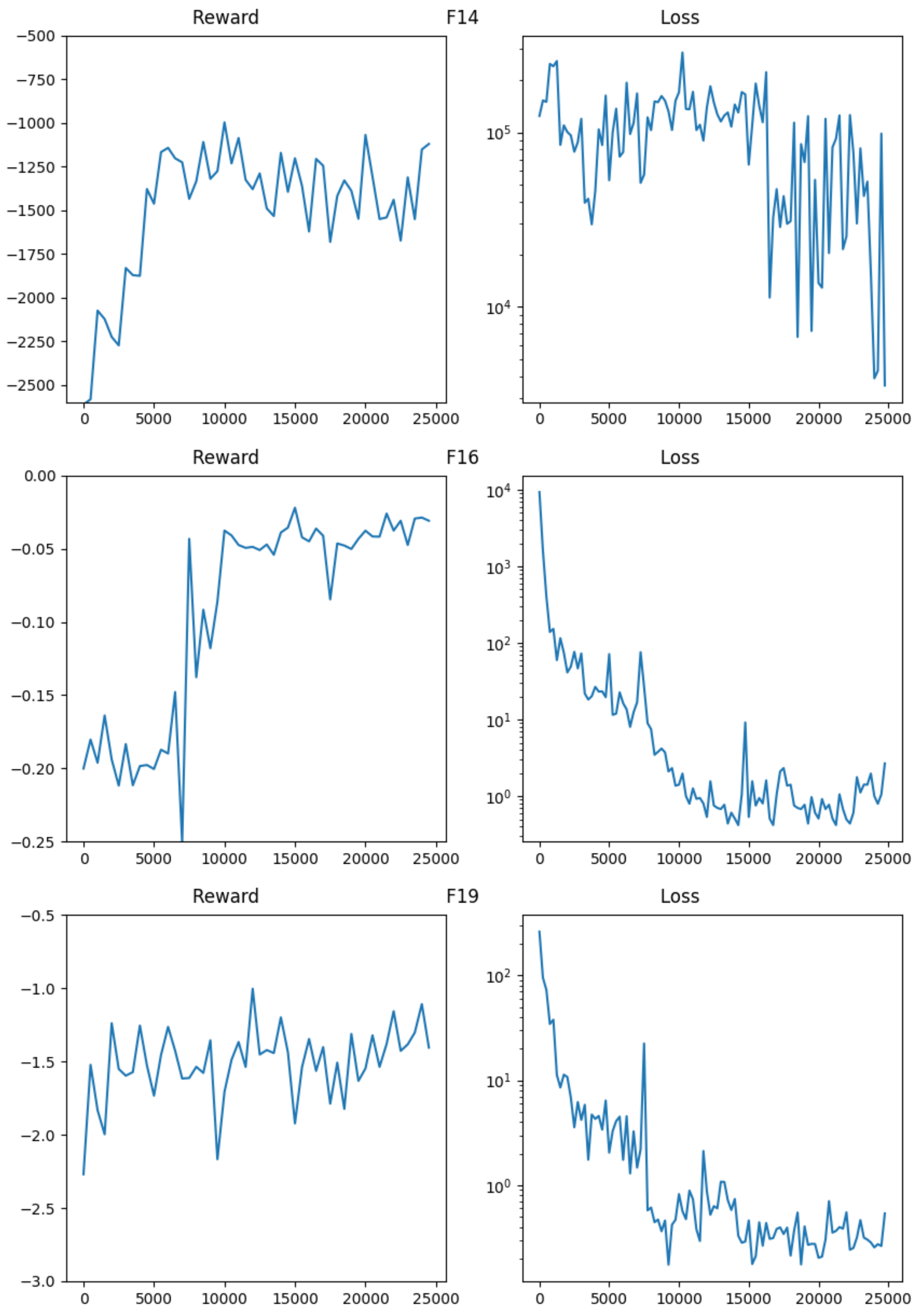

FIGURE 4.6: Rewards and loss for the FinalRange RL solution when trained on functions F14, F16 and F19.

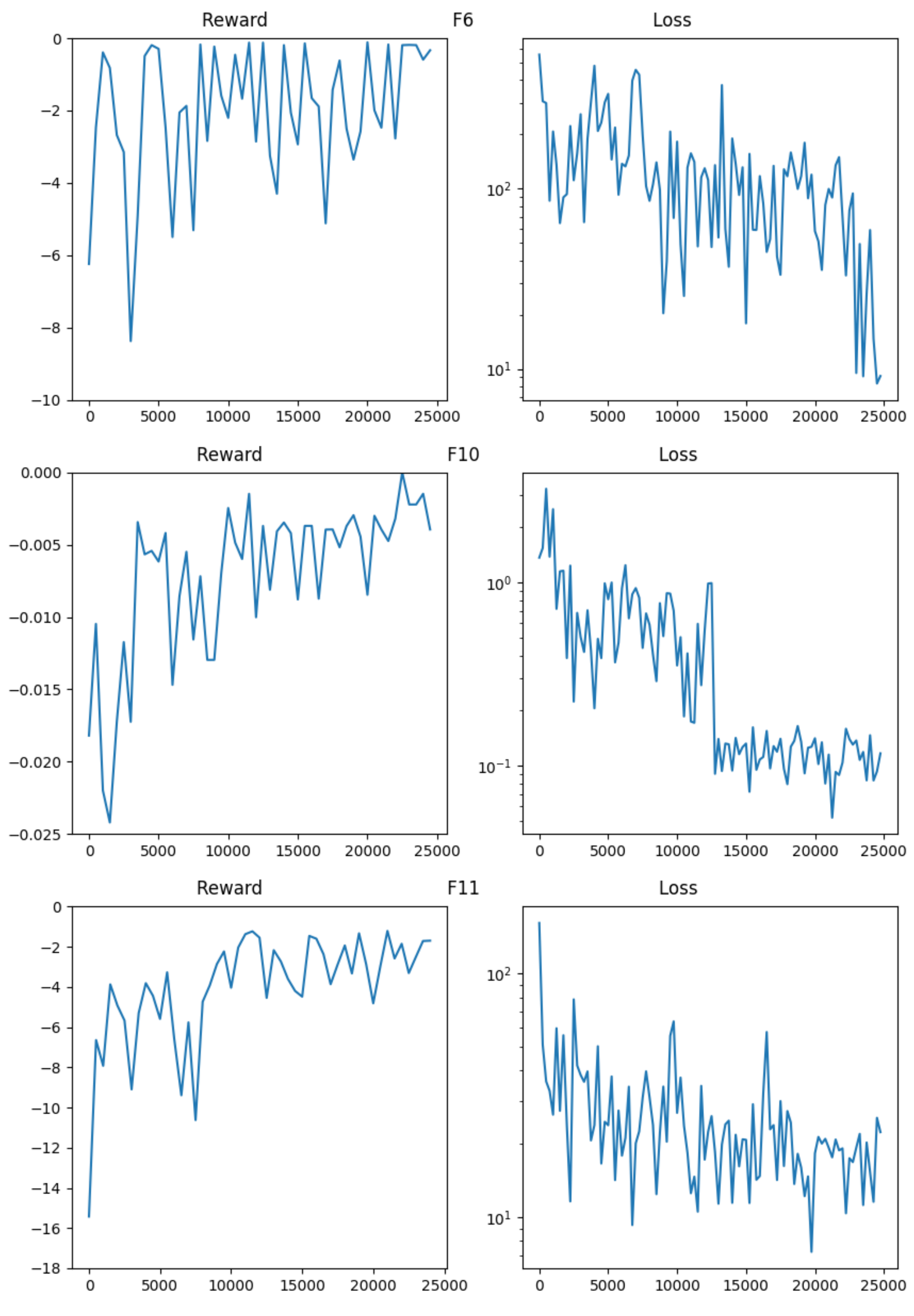

FIGURE 4.7: Rewards and loss for the FinalCombo RL solution when trained on functions F6, F9 and F11.

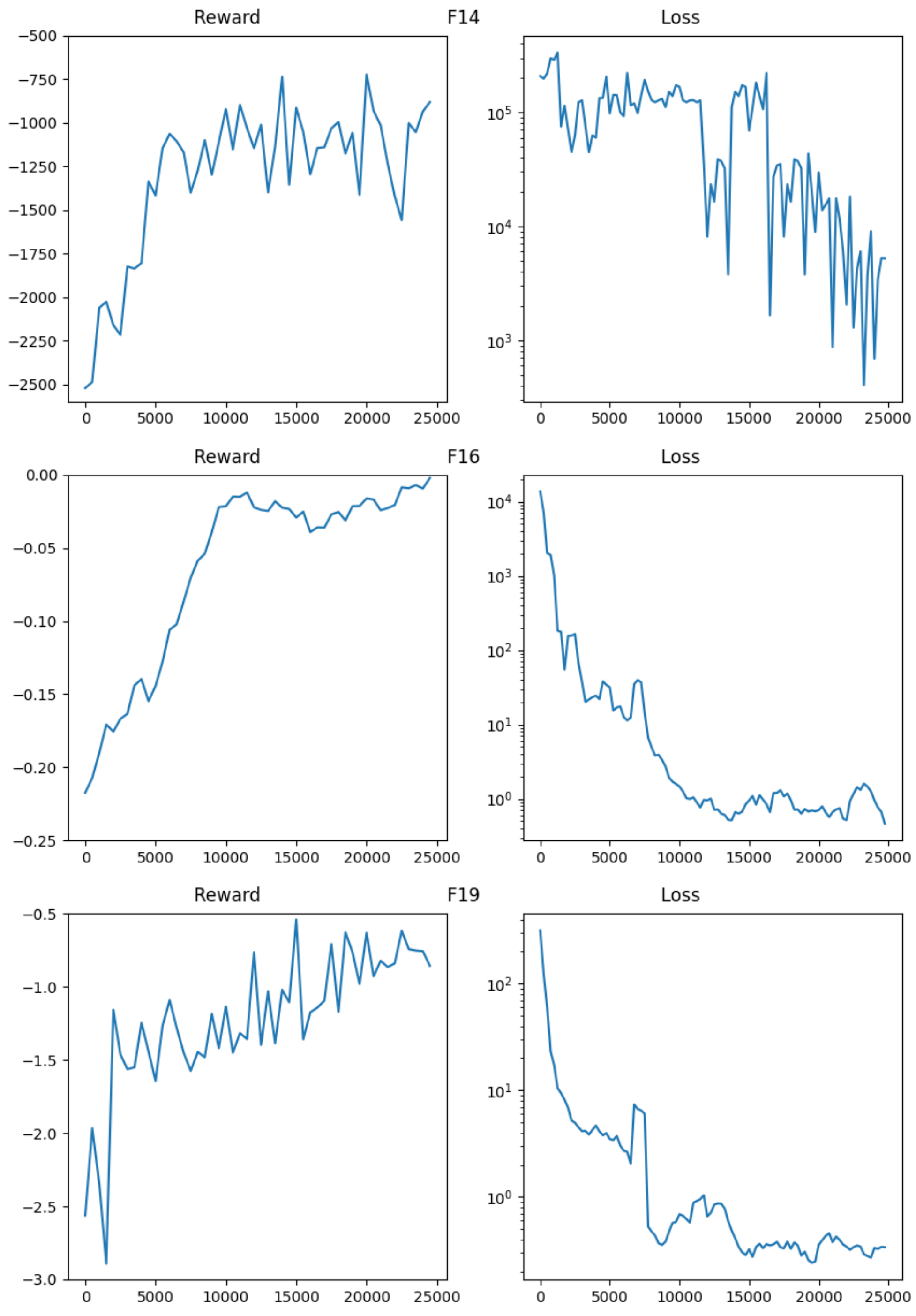

FIGURE 4.8: Rewards and loss for the FinalCombo RL solution when trained on functions F14, F16 and F19.

Chapter 5

Benchmark Performance and Generalization

The next step after finding a reinforcement learning solution that is effective at solving individual functions, is to generalize the problem to have an agent/environment learn how to effectively optimize multiple functions. By solving multiple functions we mean that the agent is not trained on a single function but on a set of many functions. This generalization will allow us to deploy the agent on any optimization function, without having to know beforehand what function will be optimized.

5.1 Reward for Multiple Functions

One challenge when using the same agent/environment for optimizing a variety of different functions is that not all functions have the error on the same order of magnitude. For example, the error for function F16 (the Katsuura function) is usually in the $(0.01, 0.3)$ range for the UES-CMA-ES hybrid; the error for function F11 (the Rastrigin function) is in the $(2, 20)$ interval, while the error for function F14 occurs usually in the $(1000, 2000)$ interval. The ranges for these errors can be found in the results reported so far in this thesis, Figs.3.3 to 4.8. Although not reported yet, we also know that for other functions like F1 (the Sphere function) and F10 (the Rotated Griewank function), the UES-CMA-ES algorithm can frequently find the global optimum (i.e., an error of 0).

The problem with these big differences in the reward values resides in the fact that it could be used by the agent to do what is known in the RL literature as "specification gaming" or "reward hacking". These terms refer to the ability of an RL model to find loopholes in the reward functions that help them accomplish a better performance without achieving the desired goal for which the environment was designed [90]. In our case specifically, we are worried that the agent could prioritize those functions that provide higher rewards in detriments of the rest of the functions.

To avoid a potential reward hacking from our agent we designed an alternative reward function which normalizes the output values. The normalization is done by dividing the reward of a given function by the median error of 50 independent executions of UES-CMA-ES for that function. Equation 5.1 shows how the reward is calculated, f_i indicates the specific benchmark function that is being optimized and $median_i$ is the median error for that function using UES-CMA-ES.

$$R_{f_i} = \begin{cases} \frac{-\text{error}_{\text{best}}}{\text{median}_i} & \text{if it's the first execution} \\ \frac{\text{error}_{\text{best}} - \text{error}_{\text{new}}}{\text{median}_i} & \text{if the new solution is better} \\ 0 & \text{if the new solution is worse} \end{cases} \tag{5.1}$$

5.2 Results for the Entire Benchmark

5.2.1 Training the Agents

In this section we present the results of the UES-CMA-ES reinforcement learning hybrid trained and tested over the entire benchmark. The first set of results aim to compare the

FinalCombo reinforcement learning solutions using the two different reward strategies: we will call standard reward (*StdRweard*) the original reward function without the normalization, and we will refer as *NormReward* to the normalized version of the rewards.

The experiment setting for training on the entire benchmark is similar to the experimental setting described in Section 4.3. Each function was optimized for 30 dimensions - to match many of the results reported in the literature. The total of number of function evaluations used was $FEs = 30 \times 10,000 = 300,000$. The environment was configured to perform 10 restarts (i.e., a restart will be performed every $30,000$ function evaluations). Two results were collected in these experiments: the rewards obtained by the agent and the loss of the neural network.

The main difference is that instead of focusing on individual functions, the environment was modified to train the agent on a variety of functions. At every reset, the environment randomly chooses one function from the benchmark and trains the agent on that function. To allow for a better generalization, this training was performed for $100,000$ steps instead of the $25,000$ used for individual functions. The training was done using Compute Canada, and it took over a month to train each model.

Figure 5.1 shows the reward and the loss over the entire benchmark for the *FinalCombo* reinforcement learning solution using the standard reward function. Figure 5.2 shows the same results using the normalized reward function instead. It is worth noticing that the results for the standard reward function are hard to interpret because the reward value varies significantly from one function to another. As a consequence, even though some overall improvement is to be expected as the training progresses, the reward values can randomly fluctuate up and down depending on the function that is being optimized. However, the loss function plots indicate that the model is indeed learning.

Results for the normalized reward function should be easier to interpret since all the functions produce a reward value in a similar range. As it can be seen in Figure 5.2, the reward indeed improves as the training progresses, however, this improvement is not as clear as it was on the individual functions. The reason for this could be the extra challenge of learning a good restart configuration not for only one individual function, but learning instead to generalize for any given function. The loss plot shows that the model manages to reduce the loss, even though with some ups and downs along the process.

Despite the overall positive trend in the rewards, the relatively high loss values can be attributed to several factors. First, the complexity and diversity of the benchmark functions result in a high variance in the training data, making it challenging for the model to fit all aspects of the data accurately. Additionally, the loss function may penalize deviations from the target more heavily in certain areas of the solution space, particularly for complex functions with intricate landscapes. This can result in higher loss values even when the model is effectively improving its reward outcomes. Another possible reason is that the model might be overfitting to specific functions within the benchmark, leading to a slower convergence in the loss function while still achieving good performance on the reward metric. It is also worth noting that in reinforcement learning, the loss function does not always directly correlate with performance improvements, as the primary objective is often to maximize cumulative reward rather than to minimize prediction error.

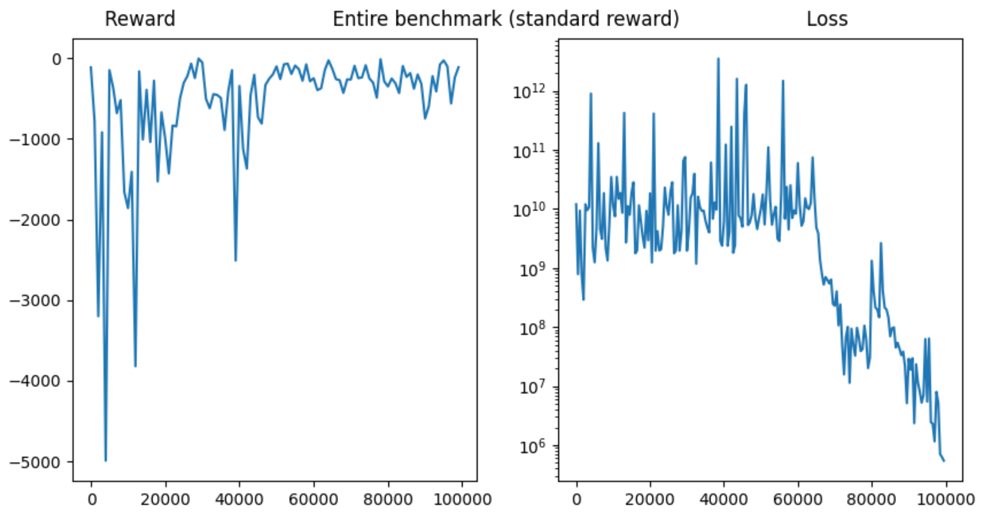

FIGURE 5.1: Rewards and loss for the FinalCombo with standard reward on the entire benchmark.

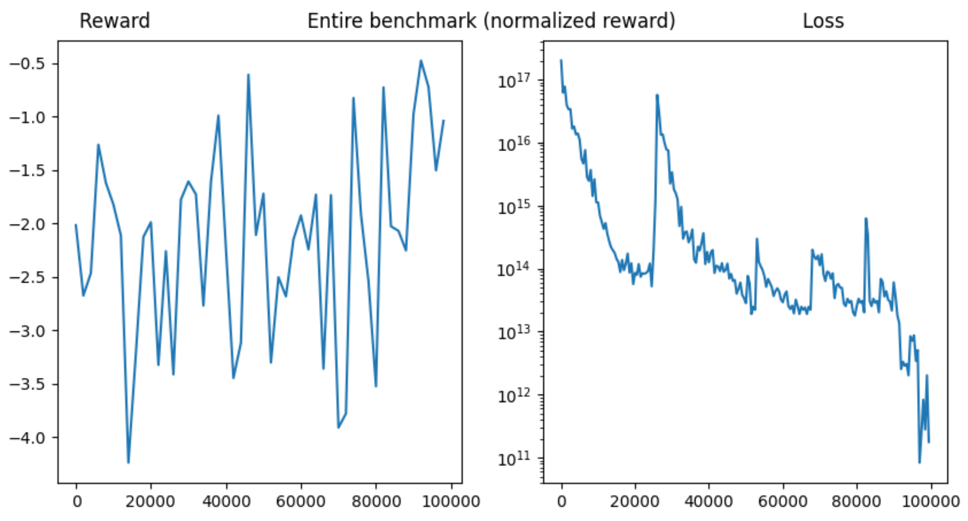

FIGURE 5.2: Rewards and loss for the FinalCombo with normalized reward on the entire benchmark.

5.2.2 Optimization Results

When the agents were trained on the individual functions, the results from the reward function plots could be easily interpreted as the error of the UES-CMA-ES hybrid on that function. However, the reward value when trained over the entire benchmark does not reveal the precise error achieved per each function. In order to precisely measure the effectiveness of the proposed RL solutions with the UES-CMA-ES hybrid, we performed a regular execution of the algorithm on the optimization benchmark. This means that we used the UES-CMA-ES hybrid with the trained models and tested it on the benchmark following the benchmark specification and the same experimental settings used in previous papers such as [63, 89, 91].

Results for the UES-CMA-ES hybrid with the Combo environment are presented in Table 5.1. The results are presented for both algorithms, one using the agent trained with the standard reward function (*Combo-StdReward*) and another using the agent trained with the normalized function *Combo-NormReward*. Both algorithms are tested on the standard CEC'2013 benchmark [41]. As describe in Section 2.1.2, this benchmark consists of a set of 28 uni-modal and multi-modal functions with various characteristics. The functions are divided in three sets: uni-modal functions (1 to 5), basic multi-modal functions (6 to 20) and composite multi-modal functions (21 to 28). Since the UES-CMA-ES hybrid has been specifically designed for optimizing multi-modal functions, the second group contains the most interesting functions in terms of topological characteristics. Results are presented for all the functions according to the experimental setup proposed for the benchmark, i.e. 51 independent trials on each function in $n = 30$ dimensions with a maximum of $300,000$ function evaluations.

Results in Table 5.1 reports the average (mean), worse (max) and best (min) over the 51 independent executions for of each algorithm on all the functions in the benchmark. Based on the average results per function, this table also reports the the relative performances $100(a - b)/\max(a, b)$ achieved by *Combo-NormReward* versus *Combo-StdReward*. These values indicate by what amount (percent) *Combo-NormReward* (b) outperforms *Combo-StdReward* (a) —- positive values indicate that *Combo-NormReward* outperforms *Combo-StdReward*; negative values indicate that the agent with the standard function performed better. A *t*-test between the two samples is also reported to allow a comparison on the basis of statistically significant differences at the 5% level.

Results in Table 5.1 show an interesting results, the agent with the standard reward provides overall better results than the agent with the normalized reward. In the set of uni-modal functions, the comparison is more balanced, with both algorithms successfully finding the global optimum in the Sphere function and having similar results on Function $\sim$ 5 with no significant statistical difference. On functions F2 and F3 the agent with the normalized reward is clearly better, while the normalized reward function leads to better results in F4. On average, the agent with the standard reward function performs slightly (12%) on this set of functions.

Results are different when we analyze the set of multi-modal functions: for a total of $\sim$ 13 out of the 15 multi-modal functions, *Combo-StdReward* has a smaller average error

TABLE 5.1: Comparison of the FinalCombo RL solution using the standard and normalized reward functions

No.	Combo-StdReward			Combo-NormReward			%-diff	*t*-test
	Mean	Max	Min	Mean	Max	Min		
1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	–	–
2	1.06E+02	5.05E+00	1.34E+03	2.44E+02	1.33E+01	6.51E+02	-56.7%	0.00
3	1.98E−02	6.19E−08	3.73E−01	1.41E+00	1.80E−07	4.10E+01	-98.6%	0.00
4	2.46E−06	0.00E+00	2.67E−05	1.88E−07	0.00E+00	1.69E−06	92.4%	0.00
5	5.40E−05	2.48E−05	8.84E−05	5.28E−05	4.02E−05	7.65E−05	2.3%	0.06
Uni-modal							-12.1%	
6	0.00E+00	0.00E+00	0.00E+00	2.64E+00	0.00E+00	2.64E+01	-100.0%	0.00
7	5.69E−02	2.78E−05	4.23E−01	1.40E−01	1.61E−03	8.13E−01	-59.5%	0.00
8	2.09E+01	2.07E+01	2.10E+01	2.09E+01	2.08E+01	2.10E+01	0.1%	0.11
9	9.82E+00	3.37E+00	2.07E+01	1.08E+01	3.99E+00	1.90E+01	-8.9%	0.05
10	0.00E+00	0.00E+00	0.00E+00	2.47E−04	0.00E+00	7.40E−03	-100.0%	0.00
11	8.62E+00	4.03E+00	1.39E+01	1.38E+01	6.84E+00	2.19E+01	-37.7%	0.01
12	7.95E+00	3.24E+00	1.10E+01	1.47E+01	6.96E+00	2.39E+01	-46.1%	0.00
13	1.11E+01	2.07E+00	3.47E+01	2.45E+01	5.13E+00	5.04E+01	-54.6%	0.00
14	1.24E+03	1.68E+02	2.35E+03	1.50E+03	7.15E+02	2.78E+03	-17.5%	0.02
15	1.03E+03	4.12E+02	1.77E+03	1.50E+03	6.67E+02	2.21E+03	-31.1%	0.00
16	2.37E−02	2.43E−03	1.99E−01	8.56E−03	2.96E−03	1.75E−02	63.8%	0.02
17	3.85E+01	3.46E+01	4.49E+01	4.61E+01	3.91E+01	5.71E+01	-16.4%	0.01
18	4.16E+01	3.47E+01	5.22E+01	4.89E+01	4.20E+01	6.72E+01	-14.9%	0.01
19	1.94E+00	8.49E−01	3.17E+00	1.96E+00	1.16E+00	2.41E+00	-0.7%	0.06
20	1.08E+01	8.62E+00	1.50E+01	1.10E+01	8.43E+00	1.50E+01	-1.6%	0.32
Multi-modal							-28.3%	
21	3.37E+02	2.00E+02	4.44E+02	3.68E+02	2.00E+02	4.44E+02	-8.7%	0.11
22	1.12E+03	4.91E+02	2.32E+03	1.51E+03	5.52E+02	2.43E+03	-25.8%	0.00
23	1.16E+03	5.79E+02	1.82E+03	1.88E+03	1.27E+03	2.82E+03	-38.4%	0.00
24	2.11E+02	2.00E+02	2.44E+02	2.07E+02	2.00E+02	2.62E+02	1.8%	0.05
25	2.68E+02	2.40E+02	3.03E+02	2.66E+02	2.42E+02	3.02E+02	0.8%	0.05
26	2.47E+02	2.00E+02	3.27E+02	2.53E+02	2.00E+02	3.31E+02	-2.2%	0.14
27	5.35E+02	3.00E+02	7.90E+02	4.69E+02	3.00E+02	7.00E+02	12.4%	0.04
28	3.00E+02	3.00E+02	3.00E+02	3.00E+02	3.00E+02	3.00E+02	0.0%	0.45
Composition							-7.5%	
Total							-19.5%	

(mean) than *Combo-NormReward*. The agent with the normalized reward function only performs better in F8 and F16. These results are statistically significant in all functions Except for F8, F9 F19 and F20. On average, the agent with the standard reward function outperforms the agent with the normalized reward function by a 28.35%.

For the set of Composition functions results are mixed, with both algorithms performing similarly. Results are statistically different for only 3 out of the 8 functions, with *Combo-StdReward* performing better in two of them F22 and F23, while *Combo-NormReward* performs better in F27. On average, over the entire benchmark, the UES-CMA-ES hybrid with the agent using the standard reward performs 19.5% better than the agent using the normalized reward.

There could be several hypotheses to explain why the DQN agent with the standard reward function is performing better than the agent with the normalized reward function in this scenario. One reason could be that the wide variance in reward values across different functions of the standard reward function is helping instead of affecting the agent. This variance might provide more exploration opportunities for the DQN agent, allowing it to learn better policies and discover new strategies for different functions. In contrast, the normalized reward function constrains the rewards to a narrower range, which could limit the agent's exploration.

Another reason could be related to the fact that information is compressed during normalization. The normalization of rewards might cause some loss of information. When rewards are normalized, the agent might not be able to distinguish between functions with different difficulty levels effectively. It could be that the agent trained with the standard reward function adapts its policies more accurately to the varying difficulty of different functions, leading to better performance. Also, normalizing rewards could make the agent overly sensitive to small variations in function error, leading to overfitting. The standard reward function, with its wider range of reward values, might be more robust to minor fluctuations, resulting in better overall performance. Future research should focus on conducting further experiments and analyses to determine which, if any, of these hypotheses are responsible for the observed performance differences.

5.2.3 Comparison against other algorithms

The next step is to compare the results of the UES-CMA-ES hybrid using the *Combo-StdReward* against some well known optimization algorithm. We would like to start by comparing our RL-hybrid algorithm against the original UES-CMA-ES hybrid and its composing algorithms, UES and CMA-ES. We have used the same implementations described in [82], and applied it to the CEC'13 benchmark using the recommended parameters from [63].

TABLE 5.2: Comparison against UES-CMA-ES, UES and CMA-ES using 300,000 function evaluations

No.	Combo-StdReward	UES-CMA-ES		UES		CMA-ES	
	Mean	Mean	%-diff	Mean	%-diff	Mean	%-diff
1	0.00E+00	0.00E+00	0.0%	1.20E−04	-100.0%	0.00E+00	0.0%
2	1.06E+02	2.13E−04	100.0%	1.49E+06	-99.9%	0.00E+00	100.0%
3	1.98E−02	6.80E+02	-100.0%	2.08E+06	-100.0%	2.21E−02	-10.3%
4	2.46E−06	0.00E+00	100.0%	1.21E+03	-100.0%	3.87E+03	-100.0%
5	5.40E−05	2.14E−07	99.6%	1.19E−02	-99.5%	0.00E+00	100.0%
Uni-modal			39.9%		-99.9%		17.9%
6	0.00E+00	5.28E+00	-100.0%	2.95E+01	-100.0%	8.41E+00	-100.0%
7	5.69E−02	7.88E−01	-92.7%	8.21E−01	-93.0%	1.59E+01	-99.6%
8	2.09E+01	2.09E+01	0.1%	2.09E+01	0.0%	2.11E+01	-0.6%
9	9.82E+00	1.20E+01	-18.1%	1.19E+01	-17.6%	1.45E+01	-32.4%
10	0.00E+00	1.91E−02	-100.0%	1.97E−01	-100.0%	1.69E−02	-100.0%
11	8.62E+00	8.17E+00	5.2%	4.48E+00	48.0%	5.47E+01	-84.2%
12	7.95E+00	8.34E+00	-4.6%	4.65E+00	41.5%	5.05E+01	-84.2%
13	1.11E+01	9.41E+00	15.4%	4.58E+00	58.8%	1.13E+02	-90.2%
14	1.24E+03	1.87E+03	-33.8%	1.92E+03	-35.6%	3.43E+03	-64.0%
15	1.03E+03	1.66E+03	-37.6%	1.63E+03	-36.8%	4.13E+03	-75.0%
16	2.37E−02	9.84E−02	-75.9%	1.39E−01	-82.9%	3.01E+00	-99.2%
17	3.85E+01	3.84E+01	0.4%	3.72E+01	3.3%	7.45E+01	-48.3%
18	4.16E+01	4.70E+01	-11.4%	4.42E+01	-5.7%	9.39E+01	-55.7%
19	1.94E+00	1.00E+00	48.5%	1.44E+00	25.7%	3.62E+00	-46.3%
20	1.08E+01	1.32E+01	-17.7%	1.22E+01	-11.1%	1.50E+01	-27.8%
Multi-modal			-28.1%		-20.3%		-67.2%
21	3.37E+02	3.56E+02	-5.3%	3.64E+02	-7.4%	3.00E+02	10.8%
22	1.12E+03	1.77E+03	-36.5%	1.84E+03	-38.9%	3.70E+03	-69.6%
23	1.16E+03	2.00E+03	-42.2%	2.02E+03	-42.7%	3.56E+03	-67.4%
24	2.11E+02	2.08E+02	1.1%	2.08E+02	1.3%	2.43E+02	-13.2%
25	2.68E+02	2.67E+02	0.3%	2.72E+02	-1.3%	2.57E+02	4.1%
26	2.47E+02	2.20E+02	10.7%	2.31E+02	6.3%	3.08E+02	-19.7%
27	5.35E+02	4.62E+02	13.7%	4.68E+02	12.6%	6.93E+02	-22.8%
28	3.00E+02	3.00E+02	0.0%	3.00E+02	-0.1%	3.00E+02	0.0%
Composition			-7.2%		-8.8%		-22.2%
Entire Benchmark			-10.0%		-31.3%		-39.1%

Table 5.2 presents the mean error achieved by each algorithm and the relative performances achieved by *Combo-StdReward* versus UES-CMA-ES, UES and CMA-ES. As in the previous table, these values indicate by what amount (percent) *Combo-StdReward* (b) outperforms UES/CMA-ES/UES-CMA-ES (a) —- positive values indicate that RL hybrid outperforms the other algorithms. A *t*-test between the two compared samples is also reported to allow a comparison on the basis of statistically significant differences at the 5% level.

For the uni-modal functions, *Combo-StdReward* exhibits significant improvements over UES and CMA-ES. The mean error is notably lower for *Combo-StdReward* in these cases, showcasing its superior performance. However, it is important to note that there are certain uni-modal functions where UES-CMA-ES performs better, suggesting that the hybridization does not uniformly benefit all problem types.

In the context of multi-modal functions, *Combo-StdReward* demonstrates a performance gain of approximately 28.17% when compared to UES-CMA-ES. The relative performance improvements over UES and CMA-ES are also noteworthy, emphasizing the effectiveness of the RL-based approach for tackling these complex functions. This results is specially relevant since UES-CMA-ES was originally designed to perform well on this type of problems, and as result in [82] show, it is state of the art in this field.

The results further reveal that *Combo-StdReward* performs competitively on composition functions within the benchmark, with slight variations in performance when compared to UES-CMA-ES, UES, and CMA-ES. This suggests that the RL-based approach maintains a similar level of performance across different problem categories.

When considering the entire benchmark, the *Combo-StdReward* hybrid consistently outperforms UES-CMA-ES, achieving an impressive overall improvement of approximately 10.04%. Furthermore, it demonstrates relative performance improvements over both UES and CMA-ES, with 31.27% and 39.15% gains, respectively. The RL-based "Combo-StdReward" hybrid outperforms UES, CMA-ES, and UES-CMA-ES on a substantial number of functions within the benchmark, achieving better results on 21 functions across the uni-modal, multi-modal, and composition function categories. This performance advantage highlights the effectiveness of the RL-based approach in a diverse set of optimization challenges

We also performed a comparative analysis between the RL-based *Combo-StdReward*, Particle Swarm Optimization, and Differential Evolution. For PSO a standard version [92] with a ring topology is used. Additional implementation details are the use of $p = 50$ particles [92], zero initial velocities [93] and "Reflect-Z" for particles that exceed the boundaries of the search space (i.e. reflecting the position back into the search space and setting the velocity to zero) [94]. Differential Evolution is an implementation of

TABLE 5.3: Comparison against PSO and DE using 300,000 function evaluations

No.	Combo-StdReward	PSO		DE	
	Mean	Mean	%-diff	Mean	%-diff
1	0.00E+00	0.00E+00	0.0%	2.60E+01	-100.0%
2	1.06E+02	4.52E+05	-99.9%	6.94E+07	-100.0%
3	1.98E−02	3.23E+06	-100.0%	5.09E+09	-100.0%
4	2.46E−06	5.92E+03	-100.0%	5.37E+04	-100.0%
5	5.40E−05	0.00E+00	100.0%	3.11E+01	-100.0%
Uni-modal			-40.0%		-100.0%
6	0.00E+00	2.42E−01	-100.0%	6.63E+01	-100.0%
7	5.69E−02	2.33E+01	-99.7%	7.58E+01	-99.9%
8	2.09E+01	2.07E+01	1.1%	2.12E+01	-1.3%
9	9.82E+00	1.53E+01	-35.7%	4.58E+01	-78.5%
10	0.00E+00	1.80E−01	-100.0%	1.76E+01	-100.0%
11	8.62E+00	2.75E+01	-68.6%	1.92E+02	-95.5%
12	7.95E+00	4.01E+01	-80.1%	2.26E+02	-96.4%
13	1.11E+01	6.84E+01	-83.7%	2.23E+02	-95.0%
14	1.24E+03	1.39E+03	-10.9%	6.67E+03	-81.4%
15	1.03E+03	1.77E+03	-41.6%	7.50E+03	-86.2%
16	2.37E−02	1.05E+00	-97.7%	3.97E+00	-99.4%
17	3.85E+01	5.34E+01	-27.8%	2.30E+02	-83.2%
18	4.16E+01	8.31E+01	-49.9%	2.58E+02	-83.8%
19	1.94E+00	2.25E+00	-13.5%	2.30E+01	-91.5%
20	1.08E+01	6.32E+00	41.6%	1.50E+01	-27.8%
Multi-modal			-51.1%		-81.3%
21	3.37E+02	2.70E+02	19.7%	3.44E+02	-2.0%
22	1.12E+03	1.48E+03	-24.2%	6.94E+03	-83.8%
23	1.16E+03	2.45E+03	-52.8%	7.77E+03	-85.1%
24	2.11E+02	2.45E+02	-13.9%	3.21E+02	-34.4%
25	2.68E+02	2.52E+02	5.9%	3.50E+02	-23.4%
26	2.47E+02	2.00E+02	19.0%	2.30E+02	6.8%
27	5.35E+02	7.09E+02	-24.4%	1.26E+03	-57.5%
28	3.00E+02	7.32E+02	-59.0%	4.08E+02	-26.4%
Composition			-16.2%		-38.2%
Entire Benchmark			-39.1%		-72.3%

DE/rand/1/bin with typical parameters of population size $p = 50$, crossover $Cr = 0.9$, and scale factor $F = 0.8$ [95]. These result are presented in Table 5.3, and were conducted using 300,000 function evaluations and 51 independent runs on each function.

In the context of uni-modal functions, *Combo-StdReward* consistently demonstrates significant improvements over both PSO and DE. The mean error for *Combo-StdReward* is substantially lower than that of PSO and DE, resulting in performance gains of approximately 40% and 100%, respectively, over these two conventional optimization techniques. This highlights the remarkable effectiveness of the RL-based approach in solving uni-modal problems.

When tackling multi-modal functions, *Combo-StdReward* continues to excel, showcasing a remarkable 51.13% improvement over PSO and a remarkable 81.36% improvement over DE. The significant performance gains over these conventional algorithms emphasize the robustness and adaptability of *Combo-StdReward* in addressing complex, multi-peaked functions.

The results for composition functions demonstrate that *Combo-StdReward* maintains a competitive edge with a performance advantage of approximately 16.21% over PSO and 38.24% over DE. These findings underscore the consistent performance of the RL-based hybrid approach, demonstrating that it can effectively optimize composite functions within the benchmark.

Considering the overall benchmark, *Combo-StdReward* exhibits a consistent and impressive performance improvement of approximately 39.17% over PSO and 72.37% over DE. These results reinforce the notion that the *Combo-StdReward* hybrid is a powerful and versatile optimization tool, capable of outperforming traditional algorithms across a wide range of optimization problems.

While it would be ideal to compare our approach against every possible metaheuristic, such as Genetic Algorithms, practical constraints necessitated focusing on a subset of well-established algorithms. PSO and DE were selected for comparison because they are among the most widely used optimization techniques in continuous domains, where our research is focused. Both methods are particularly popular due to their simplicity and effectiveness in handling a wide range of continuous optimization problems[96]. In contrast, Genetic Algorithms, while powerful, are often more associated with discrete optimization tasks and may not perform as consistently as PSO and DE in continuous domains.

One of the main motivations and hypothesis of this thesis was that using UES-CMA-ES with RL restarts would allow for improved optimization with larger budgets of function evaluations. To confirm this hypothesis we repeated the previous experiments but with

TABLE 5.4: Comparison against UES-CMA-ES, UES and CMA-ES using 900,000 function evaluations

No.	Combo-StdReward	UES-CMA-ES		UES		CMA-ES	
	Mean	Mean	%-diff	Mean	%-diff	Mean	%-diff
1	0.00E+00	0.00E+00	0.0%	1.44E−06	-100.0%	0.00E+00	0.0%
2	0.00E+00	0.00E+00	0.0%	3.78E+05	-100.0%	0.00E+00	0.0%
3	0.00E+00	1.28E−01	-100.0%	7.45E+04	-100.0%	6.80E−05	-100.0%
4	0.00E+00	0.00E+00	0.0%	1.12E+01	-100.0%	7.62E+03	-100.0%
5	0.00E+00	0.00E+00	0.0%	4.00E−03	-100.0%	0.00E+00	0.0%
Uni-modal			-20.0%		-100.0%		-40.0%
6	0.00E+00	5.28E+00	-100.0%	1.75E+01	-100.0%	1.06E+01	-100.0%
7	1.47E−02	1.97E−01	-92.5%	2.05E−01	-92.8%	1.51E+01	-99.9%
8	2.08E+01	2.09E+01	-0.2%	2.09E+01	-0.2%	2.10E+01	-0.9%
9	8.37E+00	9.87E+00	-15.2%	1.08E+01	-22.4%	1.69E+01	-50.6%
10	0.00E+00	1.19E−02	-100.0%	1.43E−02	-100.0%	1.26E−02	-100.0%
11	1.43E+00	2.22E+00	-35.8%	2.16E+00	-33.9%	4.69E+01	-97.0%
12	8.29E−01	1.06E+00	-21.9%	1.39E+00	-40.5%	4.95E+01	-98.3%
13	9.62E−01	1.72E+00	-44.1%	1.28E+00	-24.7%	1.14E+02	-99.2%
14	1.20E+03	1.72E+03	-29.9%	1.69E+03	-28.8%	4.17E+03	-71.1%
15	1.30E+03	1.30E+03	-0.0%	1.19E+03	8.1%	3.62E+03	-64.2%
16	1.39E−02	5.48E−02	-74.7%	6.08E−02	-77.2%	3.19E+00	-99.6%
17	3.35E+01	3.32E+01	1.1%	3.47E+01	-3.4%	7.42E+01	-54.8%
18	3.77E+01	3.41E+01	9.4%	3.41E+01	9.4%	9.81E+01	-61.6%
19	5.59E−02	4.31E−01	-87.0%	5.73E−01	-90.2%	3.81E+00	-98.5%
20	9.12E+00	9.22E+00	-1.0%	9.78E+00	-6.7%	1.50E+01	-39.2%
Multi-modal			-39.5%		-40.2%		-75.7%
21	3.00E+02	3.67E+02	-18.3%	3.40E+02	-11.7%	3.18E+02	-5.7%
22	1.02E+03	1.65E+03	-37.8%	1.56E+03	-34.1%	3.38E+03	-69.7%
23	1.33E+03	1.69E+03	-21.0%	1.61E+03	-17.1%	3.75E+03	-64.4%
24	2.00E+02	2.04E+02	-1.9%	2.05E+02	-2.4%	2.38E+02	-16.1%
25	2.56E+02	2.65E+02	-3.5%	2.66E+02	-3.8%	2.57E+02	-0.4%
26	2.00E+02	2.24E+02	-10.9%	2.20E+02	-9.1%	3.14E+02	-36.4%
27	3.07E+02	3.65E+02	-15.8%	4.18E+02	-26.5%	6.91E+02	-55.6%
28	2.80E+02	3.00E+02	-6.7%	3.00E+02	-6.7%	3.00E+02	-6.7%
Composition			-14.5%		-13.9%		-31.9%
Entire Benchmark			-28.9%		-43.4%		-56.8%

three times the original budget of evaluations (i.e., 900,000 function evaluations). Table 5.4 presents the comparison between *Combo-StdReward*, UES-CMA-ES, UES, and CMA-ES using 900,000 function evaluations demonstrates remarkable improvements, building on the previous findings; Table 5.5 presents the results versus DE and PSO.

In the context of uni-modal functions, Combo-StdReward exhibits a substantial performance boost when given a larger budget. The mean error is zero for all uni-modal functions, which means it finds the global optimum in every run. Although the other algorithms also improve with the larger budgets, the improvement is not so big and for some functions they miss the global optimum.

In the set of multi-modal and composition functions there is also a significant improvement in performance with the larger budget of evaluations. With an overall improvement over UES-CMA-ES of 28.85%, and even larger improvements over UES and CMA-ES. This emphasizes that the RL-based *Combo-StdReward* approach effectively capitalizes on increased function evaluations to enhance the optimization process, a capability that is largely absent in conventional metaheuristics with controlled convergence schedules.

When compared against classic algorithms such as DE and PSO, the relative improvement in all categories also increases. This confirms that our proposed algorithm is better at taking advantage of long execution budgets than those well known metaheuristics. These results suggest that the RL-based hybrid approach excels not only in terms of optimization quality but also in its ability to scale effectively with larger budgets, making it a powerful and versatile choice for addressing a wide spectrum of optimization challenges.

To rigorously evaluate the performance differences between the algorithms, a *t*-test was conducted on the error values obtained from the independent runs for each algorithm. The *t*-test is a statistical tool used to determine whether the means of two datasets are significantly different from each other. In this context, the test helps to assess whether the observed differences in the errors reported by the algorithms are due to random variation or if they reflect a true difference in performance.

The goal of this analysis is to determine if the error samples from different algorithms follow different distributions, which would indicate that the results can be considered statistically distinct. A p-value resulting from the *t*-test is used to make this determination. Specifically, if the p-value is less than 0.05 (the 5% significance level), we can reject the null hypothesis and conclude that the differences between the algorithms are statistically significant.

The p-values for each comparison are reported in Table 5.6. This table includes the p-values associated with the comparisons of the RL-based *Combo-StdReward* algorithm against the other algorithms (UES-CMA-ES, UES, CMA-ES, DE, and PSO) under two

TABLE 5.5: Comparison against PSO and DE using 900,000 function evaluations

No.	Combo-StdReward	PSO		DE	
	Mean	Mean	%-diff	Mean	%-diff
1	0.00E+00	0.00E+00	0.0%	2.66E+01	-100.0%
2	0.00E+00	1.56E+05	-100.0%	2.55E+07	-100.0%
3	0.00E+00	2.42E+06	-100.0%	1.27E+08	-100.0%
4	0.00E+00	4.85E+02	-100.0%	3.92E+04	-100.0%
5	0.00E+00	0.00E+00	0.0%	2.97E+01	-100.0%
Uni-modal			-60.0%		-100.0%
6	0.00E+00	6.90E−02	-100.0%	5.99E+01	-100.0%
7	1.47E−02	2.17E+01	-99.9%	4.05E+01	-99.9%
8	2.08E+01	2.06E+01	0.9%	2.12E+01	-1.6%
9	8.37E+00	1.47E+01	-43.2%	4.54E+01	-81.5%
10	0.00E+00	2.06E−01	-100.0%	1.17E+01	-100.0%
11	1.43E+00	2.55E+01	-94.4%	1.57E+02	-99.0%
12	8.29E−01	3.94E+01	-97.9%	2.07E+02	-99.6%
13	9.62E−01	6.67E+01	-98.5%	2.00E+02	-99.5%
14	1.20E+03	1.30E+03	-7.5%	6.07E+03	-80.1%
15	1.30E+03	1.70E+03	-23.8%	7.42E+03	-82.5%
16	1.39E−02	8.96E−01	-98.4%	4.03E+00	-99.6%
17	3.35E+01	4.96E+01	-32.4%	2.01E+02	-83.3%
18	3.77E+01	6.41E+01	-41.2%	2.36E+02	-84.0%
19	5.59E−02	2.15E+00	-97.4%	2.29E+01	-99.7%
20	9.12E+00	5.78E+00	36.6%	1.50E+01	-39.1%
Multi-modal			-59.8%		-83.3%
21	3.00E+02	2.30E+02	23.3%	3.39E+02	-11.5%
22	1.02E+03	1.46E+03	-29.6%	6.42E+03	-84.0%
23	1.33E+03	2.28E+03	-41.4%	7.57E+03	-82.3%
24	2.00E+02	2.43E+02	-17.5%	3.19E+02	-37.2%
25	2.56E+02	2.50E+02	2.2%	3.49E+02	-26.7%
26	2.00E+02	2.00E+02	-0.0%	2.16E+02	-7.2%
27	3.07E+02	6.86E+02	-55.2%	8.92E+02	-65.5%
28	2.80E+02	5.78E+02	-51.5%	4.43E+02	-36.7%
Composition			-21.2%		-43.9%
Entire Benchmark			-48.8%		-75.0%

different evaluation budgets: 300,000 and 900,000 function evaluations. A p-value below 0.05 in the table indicates that the difference in performance between *Combo-StdReward* and the corresponding algorithm is statistically significant at the 5% level, thereby affirming the robustness of the *Combo-StdReward* approach in achieving lower error values.

The p-values in Table 5.6 indicate the statistical significance of the differences in error distributions between the RL-based *Combo-StdReward* algorithm and the other algorithms across the CEC'13 benchmark functions. For most functions, the p-values are below 0.05, highlighting that the improvement of the RL hybrid is statistically significant.

TABLE 5.6: P-values from t-tests comparing Combo-StdReward against other algorithms for each function of the CEC'13 benchmark

No.	**UES-CMA-ES**		**UES**		**CMA-ES**		**DE**		**PSO**	
	300k	**900k**	**300k**	**900k**	**300k**	**900k**	**300k**	**900k**	**300k**	**900k**
1	–	–	0.00	0.00	–	–	0.00	0.00	–	–
2	0.00	–	0.00	0.00	0.00	–	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	–	–	0.00	0.00	0.00	–	0.00	0.00	0.00	–
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	**0.09**	**0.06**	**0.07**	**0.05**	0.04	0.03	0.02	0.01	0.01	0.00
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
17	**0.07**	**0.05**	0.00	**0.05**	**0.05**	0.01	0.00	0.00	0.03	0.00
18	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
19	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
20	**0.05**	0.04	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00
21	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
22	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
23	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
24	0.03	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25	0.04	0.01	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
26	0.00	0.00	0.00	0.00	0.00	0.00	**0.06**	0.00	0.00	0.00
27	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
28	**0.05**	0.01	**0.05**	0.01	**0.05**	0.00	0.00	0.00	0.00	0.00

This suggests that the differences in performance are not due to random variation, but rather reflect a true performance advantage of the RL-based approach.

However, there are a few functions where the p-values are 0.05 or above (highlighted in bold). In these cases, the statistical significance is not achieved, likely because the functions have a topology that leads all algorithms to converge to similar results. This convergence reduces the observable differences between the algorithms, resulting in higher p-values and indicating that the performance differences may not be statistically significant in these specific cases.

Additionally, for some functions, the p-value is omitted to indicate that the global optimum was found in every execution, leading to identical results across all runs. This results in a p-value that cannot be calculated since there is no variation in the data to compare

5.3 Discussion

The results presented in this chapter offer a resoundingly positive response to the research questions and hypotheses posed at the beginning of this study. The primary achievement is the demonstration of the feasibility of creating a deep reinforcement learning solution for intelligent restarts within the context of exploration-only exploitation-only hybrid algorithms. The trained agents, notably the *FinalCombo* agent with the standard reward function, have substantially enhanced the performance of the UES-CMA-ES algorithm. This accomplishment is particularly noteworthy, considering that UES-CMA-ES is already a state-of-the-art algorithm for optimizing complex multi-modal functions. By pushing the boundaries of its performance, the new algorithm emerges as one of the most promising optimization techniques available.

Furthermore, results in this chapter have also proven that the multi-restart strategy combined with the Deep Learning models can take advantage of large budget of function evaluations. The comparison of 300,000 and 900,000 function evaluations represents that Combo-StdReward has a better performance on the entire benchmark against all five other algorithms. Meanwhile, the results indicates that Combo-StdReward outperforms in basic multi-modal functions. It also proves that RL enhanced UES-CMA-ES hybrid is potential in more real world applications.

The comparative analysis against well-established algorithms such as PSO, DE, UES, and CMA-ES further illuminates the competitive edge of the DRL-enhanced hybrid approach. Specifically, the *FinalCombo* agent with the standard reward function consistently outperformed these traditional optimization methods across the board. This comparative advantage holds true for both the 300,000 and 900,000 function evaluation scenarios, highlighting the robustness and versatility of the DRL-enhanced approach.

The findings from this study have profound implications for tackling real-world optimization challenges. The enhanced performance and scalability of the DRL-enhanced UES-CMA-ES algorithm make it a highly promising tool for a wide range of applications, from engineering design optimizations to complex decision-making problems in uncertain environments. Its ability to efficiently navigate and optimize across diverse and complex landscapes presents a significant advancement in the field.

Chapter 6

Conclusions and Future Work

In summary, we proposed a novel hybridization algorithm to improve the metaheuristics, UES-CMA-ES. Our major contribution is modelling this algorithmic structure as a reinforcement learning problem and the agent will be tasked with determining where to restart the algorithm and/or how to modify the parameters of the search. The environment is the interaction between the metaheuristic and the objective function, and the agent will receive the information collected during the optimization process as the observations from the environment. The agent makes actions that decide where and how to perform the restart upon the environment, and improvement in the optimization results from one restart to the next is the reward obtained by the agent after each action. We took UES-CMA-ES as our metaheuristic algorithm, adapting it to have restarts and implementing it in Python because the original algorithm didn't have restarts.

After modelling the problem of restarts as a RL problem, it is important to find a good environment, that means finding the best observations, actions and reward function. In our initial approach, we employed a DQN agent and the environment that inputs the convergence curve of the optimization algorithm and outputs as action a solution in that curve to be used as the initial point for the restart. We tested the performance of the Agent + Environment as RL solution on the IEEE CEC'13 benchmark. However, this RL solution did not perform effective improvement in optimization. We set two comparative experiments that changes their actions made by agent while keeping other conditions same, but the results still had no improvements.

To find the reason of poor performance in the initial approach, we created an experimental framework for testing different environments. We tested the environment with different policies and the results show that the agent was unable to find suitable policies to improve the performance. Thus, we created three new environments to do the environment test experiment and got promising results in *FinalRange* and *FinalCombo* environments. After training the DQN agent on these two environments we found the *FinalCombo* environment performed better, so we will use the DQN agent +*FinalCombo* environment as our reinforcement learning solution in the research from now. Then, we did generalization that will allow to deploy the agent on any optimization function by normalizing reward values to avoid a potential reward hacking. We trained and tested over the entire benchmark and compared the results of the UES-CMA-ES reinforcement learning hybrid with other algorithms. The results of comparison represent the improvement of UES-CMA-ES Hybrid increases in all categories relatively, and this confirms that the research hypothesis of this thesis that RL restarts improves the metaheuristics.

Moreover, the research has revealed that this multi-restart algorithm is well-equipped to leverage extensive budgets of function evaluations. This is of great significance when addressing real-world problems where computational time constraints are not artificially imposed by benchmark competitions. In such scenarios, having an algorithm that can operate for extended durations and consistently deliver improved solutions is a valuable contribution. This capability distinguishes it from many metaheuristic algorithms, which often struggle to harness the full potential of additional function evaluations without extensive parameter tuning.

While this research has provided compelling answers to key research questions, it has also uncovered intriguing avenues for future work. Several questions remain unanswered and serve as valuable directions for further investigation:

- **Normalization of Reward Function**: The observation that the normalization of the reward function did not yield improvements raises questions. Although hypotheses have been proposed, future work should delve into this area to provide a definitive explanation and explore potential enhancements.
- **Performance Discrepancies Across Functions**: The absence of any improvement in certain functions, including seemingly simple uni-modal functions, is a compelling research question. This goes beyond the well-known "no-free lunch theorem." Investigating why the RL solution and restarts fail to enhance performance on specific functions could offer insights into further improvements.
- **Diverse DRL Architectures**: While this research centered on DQN agents due to their widespread use and effectiveness, there are other DRL architectures worth exploring. The field of deep reinforcement learning is rich and rapidly evolving, with options like Proximal Policy Optimization [97], Asynchronous Advantage Actor-Critic [98] and other promising agent architectures [99]. Testing a variety of architectures could be a promising avenue for future research.
- **Environmental Modifications**: The environment created for this study enabled DQN agents to effectively learn the problem. Yet, numerous questions remain. Would different observations be more effective? Could a broader set of actions enhance performance? This line of inquiry opens the door for further exploration to refine and optimize the approach.

In conclusion, this research has not only addressed fundamental questions but also catalyzed further inquiries and ongoing work. The journey from exploring the potential of DRL-driven smart restarts to opening up novel research avenues holds promise for the continued advancement of optimization algorithms and their applications in real-world problem-solving. These results mark a significant milestone in the field of optimization, with ample opportunities for future research and innovation.

Bibliography

[1] H Paul Williams. Integer programming. In *Logic and Integer Programming*, pages 25–70. Springer, 2009.

[2] Tom V Mathew. Genetic algorithm. *Report submitted at IIT Bombay*, page 53, 2012.

[3] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[4] El-Ghazali Talbi. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Annals of Operations Research*, 240 (1):171–215, 2016.

[5] Alejandro M Hernández-Díaz, Jorge Pérez-Aracil, David Casillas-Perez, Emiliano Pereira, and Sancho Salcedo-Sanz. Hybridizing machine learning with metaheuristics for preventing convergence failures in mechanical models based on compression field theories. *Applied Soft Computing*, 130:109654, 2022.

[6] Cheng-Lung Huang, Wen-Chen Huang, Hung-Yi Chang, Yi-Chun Yeh, and Cheng-Yi Tsai. Hybridization strategies for continuous ant colony optimization and particle swarm optimization applied to data clustering. *Applied Soft Computing*, 13 (9):3864–3872, 2013.

[7] Zahra Kayhomayoon, Faezeh Babaeian, Sami Ghordoyee Milan, Naser Arya Azar, and Ronny Berndtsson. A combination of metaheuristic optimization algorithms and

machine learning methods improves the prediction of groundwater level. *Water*, 14 (5):751, 2022.

[8] El-Ghazali Talbi. Hybrid metaheuristics for multi-objective optimization. *Journal of Algorithms & Computational Technology*, 9(1):41–63, 2015.

[9] Somayeh Allahyari, Majid Salari, and Daniele Vigo. A hybrid metaheuristic algorithm for the multi-depot covering tour vehicle routing problem. *European Journal of Operational Research*, 242(3):756–768, 2015.

[10] Muhammad Sardaraz and Muhammad Tahir. A hybrid algorithm for scheduling scientific workflows in cloud computing. *IEEE Access*, 7:186137–186146, 2019. doi: 10.1109/ACCESS.2019.2961106.

[11] Ren Diao and Qiang Shen. Nature inspired feature selection meta-heuristics. *Artificial Intelligence Review*, 44:311–340, 2015.

[12] Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022.

[13] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[14] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[15] George Polya. How to solve it. *M.: Uchpedgiz*, 1961.

[16] George B Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. 1990.

[17] Jack Edmonds. Matroids and the greedy algorithm. *Mathematical programming*, 1: 127–136, 1971.

[18] El-Ghazali Talbi. *A taxonomy of metaheuristics for bi-level optimization*. Springer, 2013.

[19] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[20] Nikolaus Hansen. The cma evolution strategy: A tutorial. 2016.

[21] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[22] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[23] Christian Blum, Jakob Puchinger, Günther R Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied soft computing*, 11 (6):4135–4151, 2011.

[24] Reiner Horst, Panos M Pardalos, and Nguyen Van Thoai. *Introduction to global optimization*. Springer Science & Business Media, 2000.

[25] Leo Liberti. Introduction to global optimization. *Ecole Polytechnique*, 2008.

[26] Leticia Vel´zquez, Miguel Arg´ez, Reinaldo Sanchez, Carlos Ramirez, Miguel Hernandez IV, Matt Culbreth, and Antony Jameson. Hybrid optimization schemes for global optimization: Wing modeling of micro-aerial vehicles. In *2010 DoD High Performance Computing Modernization Program Users Group Conference*, pages 149–154, 2010. doi: 10.1109/HPCMP-UGC.2010.48.

[27] Sankalap Arora and Satvir Singh. Butterfly optimization algorithm: a novel approach for global optimization. *Soft Computing*, 23:715–734, 2019.

[28] Amir Seyyedabbasi and Farzad Kiani. Sand cat swarm optimization: A nature-inspired algorithm to solve global optimization problems. *Engineering with Computers*, pages 1–25, 2022.

[29] Kris Hauser. Learning the problem-optimum map: Analysis and application to global optimization in robotics. *IEEE Transactions on Robotics*, 33(1):141–152, 2016.

[30] Domenec Puig, Miguel Angel García, and Ling Wu. A new global optimization strategy for coordinated multi-robot exploration: Development and comparative evaluation. *Robotics and Autonomous Systems*, 59(9):635–653, 2011.

[31] G.A. Vijayalakshmi Pai. Multi-objective differential evolution based optimization of risk budgeted global asset allocation portfolios. In *2014 2nd International Symposium on Computational and Business Intelligence*, pages 17–20, 2014. doi: 10.1109/ISCBI.2014.11.

[32] Georg Mainik, Georgi Mitov, and Ludger Rüschendorf. Portfolio optimization for heavy-tailed assets: Extreme risk index vs. markowitz. *Journal of Empirical Finance*, 32:115–134, 2015.

[33] Masaya Yoshikawa and Shimohigasi Yoshiaki. Hybrid ant colony optimization for intensification and diversification. In *2010 IEEE International Conference on Information Reuse Integration*, pages 359–363, 2010. doi: 10.1109/IRI.2010.5558908.

[34] Yi Shang and B.W. Wah. Global optimization for neural network training. *Computer*, 29(3):45–54, 1996. doi: 10.1109/2.485892.

[35] J.-M. Renders and S.P. Flasse. Hybrid methods using genetic algorithms for global optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(2):243–258, 1996. doi: 10.1109/3477.485836.

[36] Jaco F Schutte and Albert A Groenwold. A study of global optimization using particle swarms. *Journal of global optimization*, 31:93–108, 2005.

[37] Noorbakhsh Amiri Golilarz, Mirpouya Mirmozaffari, Tayyebeh Asgari Gashteroodkhani, Liaqat Ali, Hamidreza Ahady Dolatsara, Azam Boskabadi, and Mohammad Yazdi. Optimized wavelet-based satellite image de-noising with multi-population differential evolution-assisted harris hawks optimization algorithm. *IEEE Access*, 8:133076–133085, 2020. doi: 10.1109/ACCESS.2020.3010127.

[38] Essam H. Houssein, Bahaa El-Din Helmy, Ahmed A. Elngar, Diaa Salama Abdelminaam, and Hassan Shaban. An improved tunicate swarm algorithm for global optimization and image segmentation. *IEEE Access*, 9:56066–56092, 2021. doi: 10.1109/ACCESS.2021.3072336.

[39] Qing Cheng, Huanfeng Shen, Liangpei Zhang, and Zhenghong Peng. Missing information reconstruction for single remote sensing images using structure-preserving global optimization. *IEEE Signal Processing Letters*, 24(8): 1163–1167, 2017. doi: 10.1109/LSP.2017.2703092.

[40] Antonio Bolufé-Röhler, Alex Coto-Santiesteban, Marta Rosa-Soto, and Stephen Chen. Minimum population search, an application to molecular docking. *GECONTEC: Revista Internacional de Gestión del Conocimiento y la Tecnología*, 2 (3), 2014.

[41] JJ Liang, BY Qu, PN Suganthan, and Alfredo G Hernández-Díaz. Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization. *Computational Intelligence Laboratory, Zhengzhou University, Technical Report*, 201212(34):281–295, 2013.

[42] El-Ghazali Talbi. Machine learning into metaheuristics: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 54(6):1–32, 2021.

[43] Artur Leandro da Costa Oliveira, André Britto, and Renê Gusmão. Machine learning enhancing metaheuristics: a systematic review. *Soft Computing*, pages 1–28, 2023.

[44] Vinícius Veloso de Melo and Alexandre Cláudio Botazzo Delbem. Investigating smart sampling as a population initialization method for differential evolution in continuous problems. *Information Sciences*, 193:36–53, 2012.

[45] Salem Fawaz Adra, Ahmed I Hamody, Ian Griffin, and Peter J Fleming. A hybrid multi-objective evolutionary algorithm using an inverse neural network for aircraft control system design. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 1–8. IEEE, 2005.

[46] Laura Calvet, Jésica de Armas, David Masip, and Angel A Juan. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics*, 15(1):261–280, 2017.

[47] Ivo Pereira, Ana Madureira, Eliana Costa e Silva, and Ajith Abraham. A hybrid metaheuristics parameter tuning approach for scheduling through racing and case-based reasoning. *Applied Sciences*, 11(8):3325, 2021.

[48] Mohamed Abd Elaziz, Diego Oliva, and Shengwu Xiong. An improved opposition-based sine cosine algorithm for global optimization. *Expert Systems with Applications*, 90:484–500, 2017.

[49] Abderrahmane Ben Kacem, Oualid Kamach, Samir Chafik, and Mohamed Ait Hammou. A hybrid algorithm to size the hospital resources in the case of a massive influx of victims. *International Journal of Electrical & Computer Engineering (2088-8708)*, 10, 2020.

[50] Bo Liu, Slawomir Koziel, and Qingfu Zhang. A multi-fidelity surrogate-model-assisted evolutionary algorithm for computationally expensive optimization problems. *Journal of computational science*, 12:28–37, 2016.

[51] Ronghao Yang, Yunpeng Zhao, Haipeng Si, Zhigang Li, Ruizheng Wang, and Kai Peng. Improved population initialization method and its application in bridge optimization. In *IOP Conference Series: Earth and Environmental Science*, volume 446, page 052013. IOP Publishing, 2020.

[52] Gisbert Schneider. Virtual screening: an endless staircase? *Nature Reviews Drug Discovery*, 9(4):273–276, 2010.

[53] Marco Bernardi, Silvia Carucci, Fabio Faiola, Federico Egidi, Claudio Marini, Vincenzo Castellano, and Marcello Faina. Physical fitness evaluation of paralympic winter sports sitting athletes. *Clinical Journal of Sport Medicine*, 22(1):26–30, 2012.

[54] Chengjun Fu. Physical fitness evaluation system for athlete selection based on big data technology. *International Journal of Embedded Systems*, 15(3):249–258, 2022.

[55] Antonio Bolufé-Röhler and Dania Tamayo-Vera. Machine learning based metaheuristic hybrids for S-box optimization. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–14, 2020.

[56] Zenab Mohamed Elgamal, Norizan Mohd Yasin, Aznul Qalid Md Sabri, Rami Sihwail, Mohammad Tubishat, and Hazim Jarrah. Improved equilibrium optimization algorithm using elite opposition-based learning and new local search strategy for feature selection in medical datasets. *Computation*, 9(6):68, 2021.

[57] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific reports*, 9(1): 10752, 2019.

[58] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[59] Xiaoyan Sun, Dunwei Gong, Yaochu Jin, and Shanshan Chen. A new surrogate-assisted interactive genetic algorithm with weighted semisupervised learning. *IEEE Transactions on Cybernetics*, 43(2):685–698, 2013.

[60] Maria Chernigovskaya, Andrey Kharitonov, and Klaus Turowski. A recent publications survey on reinforcement learning for selecting parameters of meta-heuristic and machine learning algorithms. In *CLOSER*, pages 236–243, 2023.

[61] Stephen Chen, Antonio Bolufé-Röhler, James Montgomery, Wenxuan Zhang, and Tim Hendtlass. Using average-fitness based selection to combat the curse of dimensionality. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2022.

[62] Stephen Chen, Antonio Bolufé-Röhler, James Montgomery, Dania Tamayo-Vera, and Tim Hendtlass. Measuring the effects of increasing dimensionality on fitness-based selection and failed exploration. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2022.

[63] Antonio Bolufé-Röhler and Stephen Chen. A multi-population exploration-only exploitation-only hybrid on CEC-2020 single objective bound constrained problems. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

[64] Antonio Bolufé-Röhler and Stephen Chen. Minimum population search - Lessons from building a heuristic technique with two population members. In *2013 IEEE Congress on Evolutionary Computation*, pages 2061–2068, 2013. doi: 10.1109/CEC.2013.6557812.

[65] Antonio Bolufé-Röhler and Stephen Chen. Minimum population search–a scalable metaheuristic for multi-modal problems. *Investigación Operacional*, 36(1), 2015.

[66] Antonio Bolufé-Röhler and Stephen Chen. Extending minimum population search towards large scale global optimization. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 845–852. IEEE, 2014.

[67] Stephen Chen and James Montgomery. A simple strategy to maintain diversity and reduce crowding in particle swarm optimization. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 811–812, 2011.

[68] Antonio Bolufé-Röhler, Stephen Chen, and Dania Tamayo-Vera. An analysis of minimum population search on large scale global optimization. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 1228–1235. IEEE, 2019.

[69] Yasser Gonzalez-Fernandez and Stephen Chen. Leaders and followers—a new metaheuristic to avoid the bias of accumulated information. In *2015 IEEE congress on evolutionary computation (CEC)*, pages 776–783. IEEE, 2015.

[70] Stephen Chen, James Montgomery, Antonio Bolufé-Röhler, and Yasser Gonzalez-Fernandez. A review of thresheld convergence. *GECONTEC: Revista Internacional de Gestión del Conocimiento y la Tecnología*, 3(1), 2015.

[71] Stephen Chen, Shanshan Lao, and Irene Moser. Footprinting the behaviour of particle swarm optimization with increasing dimensionality. In *2023 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pages 1–6. IEEE, 2023.

[72] Helena Ramalhinho Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. *Handbook of metaheuristics*, pages 129–168, 2019.

[73] Hande Öztop, M Fatih Tasgetiren, Levent Kandiller, and Quan-Ke Pan. Metaheuristics with restart and learning mechanisms for the no-idle flowshop

scheduling problem with makespan criterion. *Computers & Operations Research*, 138:105616, 2022.

[74] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep q-learning. In *Learning for dynamics and control*, pages 486–489. PMLR, 2020.

[75] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

[76] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[77] Bowen Xu. UES-CMAES with RL. https://github.com/bxu0110/experiments/tree/main, 2024.

[78] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[79] Antonio Bolufé-Röhler, Dania Tamayo-Vera, and Stephen Chen. An LaF-CMAES hybrid for optimization in multi-modal search spaces. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 757–764. IEEE, 2017.

[80] A. Bolufé-Röhler and S. Chen. Minimum population search – a scalable metaheuristic for multi-modal optimization. In *Revista Investigación Operacional, vol 36(1)*, page pp. 85–95, 2015.

[81] Yasser Gonzalez-Fernandez and Stephen Chen. Leaders and followers—a new metaheuristic to avoid the bias of accumulated information. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 776–783. IEEE, 2015.

[82] Antonio Bolufé-Röhler and Dania Tamayo-Vera. An exploration-only exploitation-only hybrid for large scale global optimization. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1062–1069. IEEE, 2021.

[83] Tim Hendtlass. The ezopt optimisation framework. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 3110–3117. IEEE, 2019.

[84] Alejandro Piad-Morffis, Suilan Estévez-Velarde, Antonio Bolufé-Röhler, James Montgomery, and Stephen Chen. Evolution strategies with thresheld convergence. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 2097–2104. IEEE, 2015.

[85] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[86] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[87] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[88] Antonio Bolufé-Röhler and Jordan Luke. A data-centric machine learning approach for controlling exploration in estimation of distribution algorithms. In *2022 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pages 1–9, 2022. doi: 10.1109/IEMTRONICS55184.2022.9795756.

[89] Antonio Bolufé-Röhler and Ye Yuan. Machine learning for determining the transition point in hybrid metaheuristics. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1115–1122. IEEE, 2021.

[90] Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. *arXiv preprint arXiv:2201.03544*, 2022.

[91] Antonio Bolufé-Röhler and Wangwei Han. A data-centric approach to parameter tuning, an application to differential evolution. In *2023 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–9. IEEE, 2023.

[92] Daniel Bratton and James Kennedy. Defining a standard for particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 120–127. IEEE, 2007.

[93] Andries Engelbrecht. Particle swarm optimization: Velocity initialization. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8.

[94] Sabine Helwig, Juergen Branke, and Sanaz Mostaghim. Experimental analysis of bound handling techniques in particle swarm optimization. *IEEE Transactions on Evolutionary computation*, 17(2):259–271, 2013.

[95] James Montgomery and Stephen Chen. An analysis of the operation of differential evolution at high and low crossover rates. In *IEEE congress on evolutionary computation*, pages 1–8. IEEE, 2010.

[96] Stephen Chen, James Montgomery, and Antonio Bolufé-Röhler. Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. *Applied Intelligence*, 42:514–526, 2015.

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

[98] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

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