Meta Learning for Population-Based Algorithms in Black-box Optimization

Mémoire

Hugo Siqueira Gomes

Maîtrise en génie électrique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Meta Learning for Population-Based Algorithms in Black-box Optimization

Mémoire

Hugo Siqueira Gomes

Sous la direction de:

Christian Gagné, directeur de recherche

Resumé

Les problèmes d'optimisation apparaissent dans presque tous les domaines scientifiques. Cependant, le processus laborieux de conception d'un optimiseur approprié peut demeurer infructueux. La question la plus ambitieuse de l'optimisation est peut-être de savoir comment concevoir des optimiseurs suffisamment flexibles pour s'adapter à un grand nombre de scénarios, tout en atteignant des performances de pointe. Dans ce travail, nous visons donner une réponse potentielle à cette question en étudiant comment faire un méta-apprentissage d'optimiseurs à base de population. Nous motivons et décrivons une modélisation commune pour la plupart des algorithmes basés sur la population, qui présentent des principes d'adaptation générale. Cette structure permet de dériver un cadre de méta-apprentissage basé sur un processus de décision de Markov partiellement observable (POMDP). Notre formulation conceptuelle fournit une méthodologie générale pour apprendre l'algorithme d'optimisation lui-même, présenté comme un problème de méta-apprentissage ou d'apprentissage pour optimiser à l'aide d'ensembles de données d'analyse comparative en boîte noire, pour former des optimiseurs polyvalents efficaces. Nous estimons une fonction d'apprentissage de méta-perte basée sur les performances d'algorithmes stochastiques. Notre analyse expérimentale indique que cette nouvelle fonction de méta-perte encourage l'algorithme appris à être efficace et robuste à une convergence prématurée. En outre, nous montrons que notre approche peut modifier le comportement de recherche d'un algorithme pour s'adapter facilement à un nouveau contexte et être efficace par rapport aux algorithmes de pointe, tels que CMA-ES.

Abstract

Optimization problems appear in almost any scientific field. However, the laborious process to design a suitable optimizer may lead to an unsuccessful outcome. Perhaps the most ambitious question in optimization is how we can design optimizers that can be flexible enough to adapt to a vast number of scenarios while at the same time reaching state-of-the-art performance. In this work, we aim to give a potential answer to this question by investigating how to meta-learn population-based optimizers. We motivate and describe a common structure for most population-based algorithms, which present principles for general adaptation. This structure can derive a meta-learning framework based on a Partially observable Markov decision process (POMDP). Our conceptual formulation provides a general methodology to learn the optimizer algorithm itself, framed as a meta-learning or learning-to-optimize problem using black-box benchmarking datasets to train efficient general-purpose optimizers. We estimate a meta-loss training function based on stochastic algorithms' performance. Our experimental analysis indicates that this new meta-loss function encourages the learned algorithm to be sample efficient and robust to premature convergence. Besides, we show that our approach can alter an algorithm's search behavior to fit easily in a new context and be sample efficient compared to state-of-the-art algorithms, such as CMA-ES.

Contents

List of Figures

Acknowledgements

The scientific experience that I have acquired during this work is unmeasurable. Besides, I wish to express my thanks to all the people that supported me during this work.

First of all, I am grateful to my supervisor Christian Gagné for his advice, for leaving me the freedom to explore my topics of interest, the financial support and the opportunity to come to Canada.

I am delighted to have had the opportunity to work with excellent students and researchers who became my friends. Particularly, Azadeh Mozafari, Mohamed Abid, Rania Souidi, Changjian Shui, and Benjamin Léger. They helped me with their enthusiasm and hard work in the projects and to become a better scientist.

The time that I have spent at Université Laval would not have been the same without the team *Five* and many other friends. I would especially like to thank Alankrit Tomar, which brought great philosophical discussions and has been a good loser in chess.

This thesis would not be possible with tremendous support from Mariana Frias. I was fortunate enough to have my life partner and best friend, along with me, all this work.

Finally, this work is dedicated to my parents, José Gomes and Tríccia Siqueira, and my sister, Maitê Siqueira Gomes, for all the moments that we have shared that made me who I am today.

Introduction

Optimization applications encompass science, engineering, economics, and industry. The prevalent and growing use of optimization makes it important for students and professionals in all fields of science and technology. The purpose of optimization is to achieve the best solution to a given problem with a set of prioritized criteria or constraints. For example, if one wants to design a new material, it may use an optimization procedure to search for solutions improving its durability or decreasing its cost. There are many circumstances where no information of the objective function is available (e.g., no analytical equations or gradients), and the solver can only rely on the raw evaluations of the problem. This kind of situation is known as Black-Box Optimization (BBO) and can appear in many cases, such as hyperparameters tuning [1], robotics [2], and health care [3].

In the course of research on BBO, the procedure to find a suitable optimizer can be often standardized in different directions: one can use a generic black-box optimizer to a problem, with possibly some inefficiency in the search, or develop a new or adapt an existing algorithm, which will include specifics elements related to the nature of the problem. Although recent algorithms might be effective in many cases, it may require expert knowledge, laborious work to validate it, and often a specific implementation due to the problem's constraints. In addition, complex problems can appear, and further tuning or a new proposed search rule might be required. Developing a customized approach may lead to an efficient and better performing optimization at the cost of requiring significant involvement for developing the approach when that is possible. Perhaps the most ambitious question in optimization is how we can design optimizers that can be flexible enough to adapt to a vast number of scenarios while at the same time reaching state-of-the-art performance. In this work, we believe the answer to this question relies on learning the optimizer instead of figuring out each piece of a robust and effective optimizer. Therefore, there is a desire to replace hand-engineered optimizers with more effective, learned algorithms. In particular, we aim to give a potential answer to the question by investigating how to learn population-based optimizers.

Population-based algorithms [4] comprehend one of the major groups in optimization that often perform well to various types of BBO tasks because they usually do not require any property of the underlying objective function. They use a population of solutions and proceed

according to the relative evaluation of the observed values. Many applications lead to different proposed search mechanisms in this field over the past years. In addition to their parallel computing power, the benefits we get by working with population-based optimizers are that different solutions can contribute jointly in order to explore different promising regions of the objective function, which can be more effective than single-solution based search for many cases. They also have shown competitive results in many complex optimization problems (e.g., training deep neural networks for machine learning tasks). Therefore, we believe that population-based algorithms possess the adaptability required to generate effective and robust general optimizers.

Adjusting population-based algorithms for every single scenario is a challenging and costly task, which might be considered impractical. From this perspective, we propose to learn an optimizer that should develop its search behavior to fit easily in a new context. To this end, we rely on the concept of learning-to-optimize [5] or meta-learning. Learning-to-optimize is a learning paradigm that aims to learn the structure among previously seen tasks such that the learned prior can be combined to make generalizable optimizers. Although recent works have proposed different approaches to learn to optimize, there are still many open questions and no unified methodology. In this work, we train a learned population-based optimizer from scratch in various BBO scenarios in our meta-learning framework. A better understanding of methods to generate learning optimizers may improve the possibility of creating high-performance search mechanisms for a given context.

Thesis Outline

Chapter 1 introduces the field of black-box optimization and explains the major state-of-the-art algorithms. It proceeds with a description of a revised general template for population-based algorithms that serves as the basis of the remainder of the thesis. Finally, it exemplifies the template in different population-based approaches: genetic algorithms, evolution strategies, estimation of distribution algorithms, ant colony optimization algorithms, and particle swarm optimization.

Chapter 2 defines the meta-learning problem setup and describes several learning-to-optimize approaches. Then, MDP and POMDP are reviewed to introduce the proposed meta-learning framework. We reframe the learning-to-optimize problem as a policy search in a particular POMDP to learn population-based algorithms. Finally, we explain its dynamics, the performance measurement of learnable optimizers, and the framework's task distributions.

Chapter 3 concludes with implementation details of the policy architecture and how we optimize it. Then, we look at the experimental analysis in different optimization tasks. We compare the performance of our learned optimizer with two baselines, showing competitive results.

Chapter 1

Revisiting Handcrafted Population-Based Search

One of the tough challenges for all researchers in population-based optimization is to have an unified view of each algorithm and how they relate to each other. Although the main goal of this chapter is not to address this problem so far lacking in the scientific literature, we aim to provide a general structure that will be used to define our learning to optimize framework for those algorithms (Chapter 2). First, we start with a description of core elements of black-box optimization problems and introduce a general template of population-based optimizer. Then, we discuss different types of those algorithms on this template.

1.1 Black-Box Optimization

Optimization is a research field that studies the mathematical problem of finding the best solution of a given function, possibly subject to some constraints. There are countless research areas where it appears: health care accessibility [6], engineering [7] and energy management [8]. In particular, optimization is the backbone for the machine learning community since it is used for parameters optimization and hyperparameters tuning [9].

There are three components with specific roles on a optimization problem:

- An **Optimizer** that provides a candidate solution at each timestep of the optimization.
- An **Objective Function** that gives an interface to evaluate candidate solutions.
- A **Search Space** that is the space of all feasible candidate solutions.

Formally, an optimization problem is to find the best solution x^* in a search space Ω and is represented in the following way:

$$x^* = \arg\min_{x \in \Omega} f(x) \tag{1.1}$$

where $f(.)$ is a *cost function* that we would like to minimize at a reasonable time. An equivalent formulation describes the maximization case, where the *candidate* x attempts to maximize $f(.)$, in which case the *cost function* is referred as *objective function*.

We now introduce useful distinctions and other particular cases of different optimization problems to build a definite terminology in the next chapters and restrict which type of optimization problem is useful for our case.

Search Space

In *Continuous Optimization* [10], the candidates solutions are within a range of values (e.g. real numbers). Therefore, the search space Ω might be a subset of $\mathbb{R}^d$, where d is the dimension of the search space. In contrast, *Discrete Optimization* [11] defines Ω as a set of finitely elements (e.g., binaries or integers numbers). There are hybrid cases where part of the solution x is continuous and another part is discrete (e.g. mixed integer programming [12]).

Constrained Optimization [13] considers the problem where the search space is bounded by a collection of inequalities which defines a *feasible* search space. Conversely, *unconstrained optimization* [14] have no restrictions on the variable values. Even though algorithms are often designed specifically for a constrained scenario, one could simply reformulate the objective function to include a *penalty term* if the solution is away from the feasible search space. Therefore, approaches are often easily modified to each version of the given problem.

Objective Function

Optimization problems can be different concerning the properties of the objective functions. A particular case appears in the *black-box optimization problem*. We assume that $f(.)$ is *unknown*, which means that it can not be formulated as a closed-form equation, and we can only query a *candidate solution* x to receive its evaluation through some real-world experiment (e.g., robotics or simulation code). Alternatively, this field is also known as *derivative-free optimization* since many optimizers can not rely on the derivative of the function to build a gradient direction.

Definition 1.1.1 (Black-box Function)**.** The objective function $f(.)$ is a black-box function iif no assumption of f is available and it can only provide an evaluation $f(x)$ when a candidate x is queried during the optimization procedure.

We now describe three classification dimensions for different objective functions: the number of optimal points, the number of objectives, and noise evaluation.

Local vs Global Optimizers

There are some cases where the objective function is *multi-modal*, i.e., there are multiple local optimal candidates. They are known as *local minimum*.

Definition 1.1.2 (Local Minimum)**.** Let x be a candidate solution for $f(.)$. The candidate $x^* \in \Omega$ is a *local minimum* of $f(.)$ if $\forall x \in I, f(x^*) \leq f(x)$, where I is an interval of the domain.

Although there are many optimization problems in which returning a local minimum is enough [15], one might need to design the optimizer to find for a *global minimum* candidate in order to optimize the objective function.

Definition 1.1.3 (Global Minimum)**.** Let x be a candidate solution for $f(.)$. The candidate $x^* \in S$ is a *global minimum* of $f(.)$ if and only if $\forall x \in \Omega, f(x^*) \leq f(x)$, where S is the domain of $f(.)$.

Therefore, we distinguish *global optimization methods* as algorithms that attempt to find the global optimum x^* of the objective function while *local optimization methods* settle a search for a reasonably good local optimum.

Single vs Multi-objective Optimization

We can also classify the optimization problem by the number of objectives functions. The most common optimization problems have a single objective function. However, the goal of some problems might be to optimize the trade-off between multiple objectives. This is known as *multi-objective optimization* [16]. Under this setting, we want to minimize multiple objective functions of the same solution space simultaneously. In practice, we can reformulate the multi-objective optimization function as a weighted combination of all different objectives. Another class of problems is known as *feasibility problems*, where it has no objective functions, and the goal is to find values for the variables that satisfy the constraints of the problems.

Noisy vs Non-noisy Functions

In many real-world scenarios, it is plausible that the observed evaluation of the objective function $f(.)$ at the candidate x contains corrupted random noise measurements. We can consider that there is an underlying noise evaluation. Thus, multiple evaluations on f lead to different results, and we often define the new objective function as:

$$g(x) = f(x) + \epsilon \tag{1.2}$$

where the noise ϵ is (without loss of generality) assumed to be additive, and often following a zero-mean gaussian distribution $\epsilon \sim \mathcal{N}(0, \sigma)$.

State-of-the-Art Approaches

In recent literature, two classes of algorithms emerge as a robust and versatile solution to BBO: Bayesian Optimization (BO) and Evolutionary Algorithms (EAs).

- **Bayesian Optimization** [17] uses a statistical model (e.g., Gaussian Processes) which provides a posterior probability distribution for the objective function at a candidate point x. Each step of the iteration, the algorithm updates the model (i.e., the posterior) and optimizes the acquisition function, which measures the possible value of $f(x)$ at a new point x. Then, the algorithm can select the next promising candidate. BO is known to be sample efficient, and other variants can use prior knowledge to improve its performance.
- **Evolutionary Algorithms** [18] are a class of population-based algorithms. They start with a set of search points (known as candidate solutions), usually generated randomly, that iteratively change by evolutionary operators. Each operator eliminates or moves search points in the domain to achieve higher objective values. EAs are known to provide more diverse candidates [19], but it usually requires more evaluations than Bayesian Optimization.

Final Remarks

For the remainder of this work, we will consider the problem of global optimization of a single black-box objective function without noise and zeroth-order feedback. Specifically, we no longer refer to *optimization* as discrete, constrained, nor discrete search space (if not specified).

In this work, we aim to make a contribution to black-box optimization to become an actual end-user product (reliable, flexible, and efficient). We suggest that learning prior knowledge of related optimization tasks can automatize design choices to create new parameterized efficient population-based algorithms.

1.2 Population-Based Search

We consider a black-box optimization problem:

$$x^* = \arg\min_{x \in \mathbb{R}^d} f(x) \tag{1.3}$$

where the objective function $f : \mathcal{X} \to \mathbb{R}$ can be evaluated at each point $x \in \mathcal{X}$ and no other information about f is available (e.g. analytical form, gradient).

In this problem, an iterative optimizer outputs a time-ordered sequence of evaluations:

$$f(x_1), f(x_2), \ldots, f(x_t) \tag{1.4}$$

where t is the time of each evaluation step done by the optimizer for a given problem f and each point $x_1, x_2, ..., x_t$ is generated by a decision-making process of the optimizer. When faced with a new problem, one might have two options: a population-based search or a single-solution based search. To simplify, we assume the difference between them as how the optimizer generates a trajectory of evaluations. In the case of population-based search, it usually evaluates the target function using a batch of points (known as *population*) while single-solution based algorithms manipulate and transform a single solution during the search. Indeed, many proposed search mechanisms are similar between them, and we can also consider some hybrid approaches. We refer λ as the *population size* and if $\lambda = 1$ we have a single-solution based search and if $\lambda > 1$ we have a population-based search. Thus, we can rewrite Equation 1.4 for both cases as:

$$\overbrace{\{f(x_1), \ldots, f(x_\lambda)\}_0}^{\lambda \geq 1}, \overbrace{\{f(x_1), \ldots, f(x_\lambda)\}_1}^{\lambda \geq 1}, \ldots, \overbrace{\{f(x_1), \ldots, f(x_\lambda)\}_t}^{\lambda \geq 1} \tag{1.5}$$

Let $P^t = \{P_X^t, P_Y^t\}$ be the population at time step t where $P_X^t = \{x_0, \ldots, x_\lambda\}_t$ is the set of points on the search space and $P_Y^t = \{f(x_0), \ldots, f(x_\lambda)\}_t$ is the set of function evaluations at those points. Therefore, we can describe a general *optimization rollout* as a sequence of search points and evaluations over time:

$$P_X^0, P_Y^0, P_X^1, P_Y^1, \ldots, P_X^T, P_Y^T. \tag{1.6}$$

Specifically, a population-based search approach is described in Algorithm 1. In population-based optimization algorithms, we have a generic update formula to the population at each time step:

$$P_X^t \sim \pi(P_X | \{P_X^i, P_Y^i\}_{i=0}^{t-1}), \tag{1.7}$$

where π defines a distribution of probability over different search points P_X; and $\{P_X^i, P_Y^i\}_{i=0}^{t-1}$ is the *optimization rollout* at time $t-1$ which is equivalent of all search points and the evaluations done until time t. The optimization starts with the first set of points P_X^0 (known as *initial population*) (line 2). Then, the function is evaluated at those points (line 3). The optimizer receives the respective evaluations and makes the decision process to generate the next batch of points to be evaluated (line 5). The procedure continues until a *stopping criteria* (e.g. maximum budget) is reached (line 4).

Algorithm 1 General Template of Population-Search Algorithms

procedure OPTIMIZE(f)
  $P_X^0 \leftarrow$ generate initial population
  $P_Y^0 = f(P_X^0)$
  for t = 1, 2, ... **do**
    $P_X^t \sim \pi(P_X | \{P_X^i, P_Y^i\}_{i=0}^{t-1})$
    $P_Y^t = f(P_X^t)$
  end for
end procedure

In this context, different optimizers can be defined by hand-designed heuristics and search mechanisms that are represented by π. A unified view and a systematic and theoretical analysis of population-based search mechanisms are found in [20]. In the next subsections, we attempt to achieve a comprehensive and consolidated view of different instances of population-based optimizers that we will use to support our meta-learning framework (Chapter 2). We are particularly interested in seeing how the distribution π relates to each search mechanism explored in two major fields of population-based algorithms: Evolutionary Algorithms (EAs) and Swarm Intelligence (SI).

1.3 Evolutionary Algorithms

Evolutionary Computation is an undergoing research field that recently had increased interest in the deep learning community [21; 22]. Although the roots of this field can be traced back to the 1950s, the unification of the concepts [18], methods [20] and evaluation procedures [23] of the field is still seldom studied.

Evolutionary System

We start by defining what constitutes an evolutionary system. In the scope of this work, similar to [18], an evolutionary system is a term borrowed from biology that refers to a Darwinian evolutionary system. In this type of system, we usually have the following components:

- a **population of individuals** competing for limited resources.
- a **time-discrete operator** that changes this population over time due to birth, reproduction, and death of the individuals.
- a concept of **fitness evaluation** that describes the ability of the individual to survive and reproduce.

There are many variants of this system. One could have more than one population or have multi-population evaluation [24]. It is worth mentioning that other biology ideas (e.g., Lamarckian properties [25] and Baldwin Effect [26]) can also be implemented on an evolutionary system. Although these changes can play a great role in the evolutionary computation community, it would be out of the scope to discuss them in detail. An in-depth explanation can be found in [18].

We need to use those components' definition to elucidate our view of an evolutionary system into an algorithmic perspective to solve a given optimization problem. To this end, we assume that we have a *black-box function* (fitness function) that can be evaluated at a given candidate solution x (individual). Mathematically, one can view individuals in the population as vectors $x^d \in \mathbb{R}^d$, which are points in the search space that provide information about the landscape of the fitness function. Then, we apply a set of evolutionary operators that changes these sample points to a new set of points (known as *offspring*), which represents the next batch of points evaluated on the fitness function. These evolutionary dynamics produce a fitness-biased exploration of the search space. When the procedure has reached a time-limited budget (or any other stopping criteria), the evolutionary algorithm's result is the best point found during the search.

In this view, every evolutionary algorithm defines a trajectory over time through a complex evolutionary population space. Consequently, the typical structure of all evolutionary algorithms is that, at each time step, the new set of evaluated points is the *offspring*, which are denoted as $P^t_{X'}$. We can see the Equation 1.6 as a *rollout* of the offsprings:

$$P^0_{X'}, P^0_{Y'}, P^1_{X'}, P^1_{Y'}, \ldots, P^T_{X'}, P^T_{Y'} \tag{1.8}$$

We now describe traditional algorithms in this category: Genetic Algorithms (GAs), Evolutions Strategies (ES), and Estimation of Distribution Algorithms (EDAs). Remember that we want to outline each algorithm according to the general distribution π (see Equation 1.7), and therefore we focus on its search components.

1.3.1 Genetic Algorithms

Genetic Algorithms (GAs) are a quite popular class of EAs. They started to be applied in the 1980s for different optimization and machine learning problems [27; 28]. In the early days of the method, each individual had a binary representation, but nowadays, different representations can be used [29]. In recent works [22; 30], GAs have demonstrated competitive results to other gradient-based approaches when training deep neural networks.

Algorithm 2 General Template of Genetic Algorithms

1: **procedure** OPTIMIZE(f)
2: $P^0_{X'}, P^0_{Y'} = \emptyset, \emptyset$
3: $P^0_X \leftarrow$ generate initial population
4: $P^0_Y = f(P^0_X)$
5: **for** t = 0, 1, 2, ... **do**
6: $P^{t+1}_X, P^{t+1}_Y = \text{Replacement}(\{P^t_X, P^t_Y\}, \{P^t_{X'}, P^t_{Y'}\})$
7: $P^{t+1}_{X'} = \text{Selection}(\{P^{t+1}_X, P^{t+1}_Y\})$
8: $P^{t+1}_{X'} = \text{Reproduction}(P^{t+1}_{X'})$
9: $P^{t+1}_{Y'} = f(P^{t+1}_{X'})$
10: **end for**
11: **end procedure**

We describe a general version of GAs in Algorithm 2. The generator π (Equation 1.7) corresponds to three core steps:

- **Replacement** of points in the current population (line 6). The replacement operator is often an evolutionary operator in which the offsprings $\{P^t_{X'}, P^t_{Y'}\}$ replaces completely the parents $\{P^t_X, P^t_Y\}$.
- **Selection** of a new population (line 7). GAs use a probabilistic selection where the absolute values of the fitness evaluations are associated with individuals. An example is the *roulette wheel selection*, where each individual is assigned with a selection probability that is proportional of its fitness. Let $P^t_{Y_i}$ be the fitness of the individual $P^t_{X_i}$ in the current population P_t. The probability of $P^t_{X_i}$ to be selected is $\frac{P^t_{Y_i}}{\sum_{j=1}^n P^t_{Y_j}}$.
- **Reproduction** of the selected population (line 8). The reproduction mechanism is usually two operators: crossover and mutation. The former is applied to two solutions to generate a new solution where each bit is chosen from either parent with equal probability (*uniform crossover*). The latter is applied to one solution that is randomly modified by flipping a bit.

1.3.2 Evolution Strategies

Evolution Strategies (ESs) is another subclass of EAs. They started to be developed in 1964 [31], and since early applications, they have shown promising results [32]. They also include a wider theory literature on convergence analysis compared to others evolutionary algorithms [33]. Unlike GAs, ESs is commonly used in continuous optimization problems where the individuals' representation is real-value vectors.

Algorithm 3 General Template of Evolution Strategies

1: **procedure** OPTIMIZE(f)
2: $P^0_{X'}, P^0_{Y'} = \emptyset, \emptyset$
3: $P^0_X \leftarrow$ generate initial population
4: $P^0_Y = f(P^0_X)$
5: **for** t = 0, 1, 2, ... **do**
6: $P^{t+1}_X, P^{t+1}_Y = \text{Replacement}(\{P^t_X, P^t_Y\}, \{P^t_{X'}, P^t_{Y'}\})$
7: $P^{t+1}_{X'} = \text{GenerateOffspring}(\{P^{t+1}_X, P^{t+1}_Y\})$
8: $P^{t+1}_{Y'} = f(P^{t+1}_{X'})$
9: **end for**
10: **end procedure**

Furthermore, different variants of ESs can be considered state-of-the-art on many benchmarking problems. CMA-ES [34], short for "Covariance Matrix Adaptation Evolution Strategy", is known for its state-of-the-art performance in derivative-free optimization. However, its performance can deteriorate as the dimensionality of the problem increases [35]. Natural Evolution Strategies (NES; [36]) is a variant that are more used in high-dimension problems related to machine learning [21].

In this work, we focus on the simplest version of this class of algorithms. We describe ES in Algorithm 3. The generator π (Equation 1.7) follows two steps:

- **Replacement** of points in the current population (line 6). They usually implement an elitist replacement, which consists of selecting the best individuals from the parents P^t_X and the offsprings $P^t_{X'}$ based on their evaluations P^t_Y and $P^t_{Y'}$.
- **GenerateOffspring** operator which generates new points to be evaluated (line 7). ESs normally uses a Gaussian distribution for mutation and does not use crossover operator. In particular, $P^{t+1}_{X'}$ is generated by sampling $\epsilon_1, \ldots, \epsilon_d \sim \mathcal{N}(0, I)$ and adding it with a predefined mutation factor σ to each member in the current population so that, for each individual i, we have $P^{t+1}_{X'_i} = P^{t+1}_{X_i} + \sigma\{\epsilon_1, \ldots, \epsilon_d\}$

1.3.3 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs) is a subclass of EAs. They make use of the idea of a probabilistic distribution that represents the current population to reproduce new offsprings [37; 38]. Instead of using a common evolutionary mechanism such as mutation or crossover, they transform the optimization problem into a search over probability distribution. Formally, the probabilistic model $Q^t(x)$ represents an explicit model of promising regions in the search space.

Algorithm 4 General Template of Estimation of Distribution Algorithms

1: **procedure** OPTIMIZE(f)
2: $P^0_{X'}, P^0_{Y'} = \emptyset, \emptyset$
3: $P^0_X \leftarrow$ generate initial population
4: $P^0_Y = f(P^0_X)$
5: $Q(x) \leftarrow$ initialize a probability model
6: **for** t = 0, 1, 2, ... **do**
7: $P^{t+1}_X, P^{t+1}_Y = \text{Selection}(\{P^t_{X'}, P^t_{Y'}\})$
8: $Q^{t+1}(x) = \text{UpdateModel}(\{P^{t+1}_X, P^{t+1}_Y\})$
9: $P^{t+1}_{X'} \sim Q^{t+1}(x)$
10: $P^{t+1}_{Y'} = f(P^{t+1}_{X'})$
11: **end for**
12: **end procedure**

The different versions of algorithms in this class vary by the probabilistic model choice and how it is updated. Some of the examples are: Population-based Incremental Learning (PBIL; [39]), Univariate Marginal Distribution Algorithm (UMDA; [38]) and Bayesian Optimization Algorithm (BOA; [40]).

We describe a high-level version of EDAs in Algorithm 4. We initialize a generic probabilistic model based on the initial population (line 5). At each step t, we select the new population $\{P^{t+1}_X, P^{t+1}_Y\}$ as all generated points or just the most promising ones (line 7). Then, we update the model $Q^{t+1}(x)$ using the current selected population $\{P^{t+1}_X, P^{t+1}_Y\}$. Finally, we sample new offspring points $P^{t+1}_{X'}$ from the model and evaluate them (line 8).

1.4 Swarm Intelligence

Swarm Intelligence is another class of algorithms that also dominates the field of population-based search. They are usually nature-inspired to represent the collective behavior between organisms (e.g., ants, birds, wasps) [41; 42]. The extensive literature on this class of algorithms results in many applications, and an interested reader is advised to consider an extensive survey of different implementations of these algorithms [43; 44].

Different from EAs, the concept of *offsprings* usually does not exist in this context. We now refer to the search points as *particles* that do movements in the search space based on cooperation by an indirect communication mechanism. In this view, every swarm intelligence algorithm shows a trajectory over time for each particle. Therefore, the general formulation of π defines each particle's next position, given all previous particles' history. We choose to describe Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO), which are among the most successful Swarm Intelligence-inspired Optimization Algorithms.

1.4.1 Ant Colony Optimization Algorithms

Ant Colony Optimization Algorithms (ACO) were first proposed in [45]. They have achieved strong results in an extensive list of problems [46] and are often used for combinatorial optimization (e.g., routing and scheduling) [47]. In addition, they have been extended to deal with continuous optimization problems [48; 49].

ACO is inspired by the cooperative foraging behavior of real-world ants seeking a path between their colony and a food source. This class of algorithms aims to mimic the principle of using a volatile chemical trail (pheromone) that is left on the ground by ants as a way to communicate with each other. Indeed, the pheromone trail memorizes the characteristics of promising generated solutions, which will guide other ants towards the best points in the search space.

Algorithm 5 General Template of ACO in Continuous Space

```
1:  procedure OPTIMIZE(f)
2:      P_X^0 ← generate initial population
3:      P_Y^0 = f(P_X^0)
4:      Q(x) ← initialize the distribution of pheronome trails
5:      for t = 0, 1, 2, ... do
6:          Update pheromone trails
7:            Evaporation()
8:            Reinforcement()
9:          P_X^{t+1} = SolutionConstruction(Q(x))
10:         P_Y^{t+1} = f(P_X^{t+1})
11:     end for
12: end procedure
```

Based on this idea, a generic template is displayed in Algorithm 5. First, the pheromone information is initialized (line 4). The algorithm is then mainly composed of two iterative steps:

- The **Pheromone Trail Update** (line 6-8). The role of the pheromone trail is to increase the probability of an ant select a particular path. The trail has a decreasing effect over time (evaporation process, line 7), and the quantity left on the trail depends on the fitness evaluation of the ants (reinforcement process, line 8). In continuous optimization problems, the trail is represented as a continuous probability density function.
- The **Solution Construction** (line 9) that characterizes a local heuristic definition to guide the search in addition to the pheromone. We refer the reader to [50] for further references in continuous domain.

1.4.2 Particle Swarm Optimization

In the early 1990s, several studies analyzed different animals capable of sharing information among their group. Inspired by these works, Particle Swarm Optimization Algorithms were introduced in [51]. They are based on natural organisms' social behavior (e.g., bird flocking or fish schooling) while foraging food. In its earliest days, they have been successfully designed for continuous optimization [49].

In this algorithm, a population (or *swarm*) consists of particles moving around in a D-dimensional search space. During the optimization rollout, each particle updates its position and velocity. The optimization takes advantage of the cooperation between the particles; that is, some particles' success will influence their companions' behavior.

Algorithm 6 General Template of PSO in Continuous Space

1: **procedure** OPTIMIZE(f)
2: $P_X^0 \leftarrow$ generate initial population
3: $P_Y^0 = f(P_X^0)$
4: $v^t \leftarrow$ initialize the velocity of each particle
5: **for** t = 0, 1, 2, ... **do**
6: $w, r_1, r_2 \sim \mathcal{N}(0, I)$
7: **for** all particles i **do**
8: $P_{X_i}^{t*} = \min(P_{X_i}^0, \ldots, P_{X_i}^t)$ ▷ the best position of particle i
9: $P_X^{t*} = \min(P_X^t)$ ▷ the best position among all particles
10: $v_i^t = wP_{X_i}^{t-1} + r_1(P_{X_i}^t - P_{X_i}^{t*}) + r_2(P_{X_i}^t - P_X^{t*})$
11: $P_X^{t+1} = P_X^t + v^t$
12: **end for**
13: $P_Y^{t+1} = f(P_X^{t+1})$
14: **end for**
15: **end procedure**

We represent the basic optimization model of PSO in Algorithm 6. Each particle i correspond to each member X_i of the current population P_X^t. A particle has its own position and velocity. At each iteration (line 5), the particle updates its position P_X^{t+1} based on the previous position P_X^t and the current velocity v_t (line 11). The velocity changes stochastically based on the particle's best position and the best position among all other particles (line 10).

1.5 Conclusion

In this chapter, we first formalize optimization problems and define *continuous, single-objective, without noise and, black-box* tasks as our problems of interest. Then, we revised a general template of population-based algorithms. We reasoned that any population-based optimizers generate new search points P_X^t at each time step t using a general update formula $\pi(.)$ (Equation 1.7). In the context of Evolutionary Algorithms, the points selected from $\pi(.)$ are referred

as *offsprings* while in Swarm Intelligence, they are defined as each particle's next position. In the next chapter, we will explore this underlying general structure of population-based algorithms. We will formalize the *learning to optimize* problem as a policy search problem in a particular Partially observable Markov decision process (POMDP) by parameterizing $\pi(.)$ as a deep neural network (Equation 1.7).

Chapter 2

Learning Population-Based Algorithms

As presented in the previous chapter, population-based algorithms are powerful optimizers but rely on many design choices. Unfortunately, this can require extensive expert knowledge and might lead to poor performance in new problem settings. The main contribution presented in this current thesis is to propose a general meta-learning framework to learn data-driven optimizers. We aim to provide a systematic way to replace hand-designed search mechanisms with a parametric population-based optimizer, trained on a set of related tasks in order to infer a general optimization model that can perform well for optimizing over problems of the same characteristics.

In this chapter, we will discuss a general meta-learning framework to train data-driven optimizers. We start with a brief explanation of learning to learn (meta-learning) and provide some background on Partially Observed Markov Decision Processes (POMDPs). After, we present the core elements of the proposed meta-learning framework. Finally, we show how to generate tasks and measure performance of the the trained optimizers.

2.1 Learning to Learn

Deep learning have achieved remarkable performance in a variety of applications: text classification [52], robotics [53], music generation [54] or even producing synthetic media [55]. However, each application is often restricted to train models specialized for a single task. Learning to learn is a promising framework that aims to train a model on various learning tasks, such that it can solve new learning tasks using only a small number of training samples.

In the standard meta-learning problem setup [56], our goal is to train a model with parameters θ that can adapt on different but related learning tasks. Formally, we associate each task with a dataset $\mathcal{D}$ and we assume a distribution $p(\mathcal{D})$ over different datasets. This distribution differs

for each particular context. In a supervised setting, the datasets may contain both feature vectors, and true labels [57]. In reinforcement learning, one can define each task in the dataset as a Markov Decision Process (MDP) with different reward functions [58].

Let $\mathcal{L}(\theta, \mathcal{D})$ be the performance (i.e., loss function) of the model θ on a dataset $\mathcal{D}$. The meta-learning setup is similar to a *standard* machine learning problem, but datasets are used instead of data samples. Therefore, we want to find the optimal model parameters θ^* as:

$$\theta^* = \arg\min_{\theta} \mathop{\mathbb{E}}_{\mathcal{D}\sim p(\mathcal{D})} [\mathcal{L}(\theta, \mathcal{D})] \tag{2.1}$$

There are some variants of the problem mentioned above. In Caruana et al. [59], it is explored the idea of *multitask learning*. They train a model in parallel for the tasks while using a shared representation. Another variant is known as *lifelong learning* which deals with the *catastrophic forgetting* problem of deep neural networks [60]. Recently, Ravi et al. [61] formalize an instance of the problem as *few-shot learning* which aims to train models that can achieve good performance with few labeled data (e.g., classification of a cat image with just five examples in the dataset).

In this work, we are interested in meta-learning in the context of *learnable optimizers* (i.e., learning to optimize). This research field extends the idea of meta-learning for optimization. We aim to train an optimizer over a set of related functions to achieve a general search rule that is robust and fast to optimize to new unseen functions. We can see this problem as a bilevel optimization problem [62; 63]: we consider a *base* (or *inner*) loop that uses an optimizer to solve a particular function, and the *meta* (or *outer*) loop that updates the parameters of the learned optimizer.

We can categorize recent published works into three main different groups:

- The **representation** of the learned algorithm:
  where a different parameterization for the learned optimizer have been proposed.
- The **task distribution** for meta-training:
  where different datasets for training learned optimizers have been proposed.
- The **meta-optimization methodology**:
  where the optimization procedure to train learned optimizers is studied.

Representation

The most common way to represent a learned optimizer is by using parameterized function approximators (e.g., linear combinations of features or deep neural networks). Bengio et al. [64] propose using a two-step optimization process training different networks simultaneously

on tasks while training the parameters of a general synaptic learning rule by mixing various biologically inspired learning signals. Runarsson et al. [65] have examined the use of a simple neural network as an efficient training algorithm for parameter optimization of other neural networks.

Recent works investigated more complex deep neural networks. Andrychowicz et al. [66] proposes a coordinate-wise LSTM model that outperform hand-designed algorithms on convex problems, training neural networks, and neural art tasks. They have also used *global averaging cells* (GACs) [67] to allow the network to implement L2 gradient clipping [68] and expanded the architecture using Neural Turing Machine (NTM) to simulate L-BFGS algorithms. Lv et al. [69] also considered the inputs for the representation as gradient and momentum normalized by the rolling average of gradients squared. In contrast, Metz et al. [63] use additional features such as validation loss to enable automatic regularization.

Many works explore other types of representation. They may still use neural networks but focus on learning the best hyperparameters for existing hand-designed methods. In this category, they impose a strong inductive bias making the meta-optimization procedure easier and producing better meta-generalization. Daniel et al. [70] use a linear policy mapping feature along the optimization path to log learning rate to be used with SGD, RMSProp, or momentum. Besides, Xu et al. [71] uses a simple MLP to output a scalar, which is multiplied by the previous learning rate of SGD.

In the last category, the representation relies on symbolic parameterization. We can express most optimization procedures with a small set of mathematical operations. In [72] parameterize optimizers as collections of mathematical primitives.

In this work, we propose an LSTM coupled with a Bayesian Neural Network to represent stochastic population-based optimizers. We also use a coordinate-wise architecture similar to ours [66]. This choice allows the learned optimizer to be invariant to the dimension of the optimization problem. However, the proposed meta-learning framework is not limited for this choice. Many different architectures could be tested, possibly allowing different search behaviours.

Task Distribution

As far as we know, no previous research has investigated *standard* procedure to generate common tasks for learning optimizers. In recent literature, different authors have proposed datasets that suites the specific goals in their research.

The distribution can be made of synthetic tasks to represent possible types of loss surfaces in real-world problems [73]. Another promising line of research is to build a new dataset by splitting standard datasets from other areas of machine learning. For example, Metz et al.

[74] sample different classification problems using different classes on ImageNet, which is a standard dataset in computer vision.

TaskSet [75] is a dataset specifically done for learnable optimizers, and it provides more than a thousand distinct optimization tasks commonly found in machine learning literature. However, we did not chose to use this recent dataset because it is more suitable for gradient-based approaches since each task also compute its own gradient.

In this work, we rather propose to build a dataset based on black-box functions for general stochastic optimizers from the COCO benchmark platform [76]. We can use different instances of a particular function to learn a general optimizer that learns to exploit that function's characteristics.

Meta-Optimization Methodology

We can also differ each approach based on the *meta* loop, which is responsible for updating the parameters (or configuration) of the learned optimizer. There have been numerous studies to investigate different strategies.

In the first group, one can cast learned optimizer as a sequential decision process: the base-parameter values are defined as the states while the action is the steps taken, and the reward is the final performance (i.e., loss function). Li et al. [5] use a guided policy search algorithm to train the meta-optimizer while Xu et al. [77] use Proximal Policy Optimization (PPO) and Daniel et al. [70] use Relative Entropy Policy Search [78].

Another class of methods is based on *standard* optimization algorithms: evolutionary or gradient-based algorithms. Evolutionary strategies have shown competitive results in training neural networks [21]. They often suffer from high variance, but Metz et al. [74] have shown promising results when considering the case of learned optimizer optimization. A common gradient-based approach is derived from truncated backpropagation through time. This algorithm is widely used when the learned optimizer is parameterized as LSTMs [74]. Hybrid approaches between gradient-based and evolutionary algorithms are also proposed for meta-optimization like Guided ES [79].

In this work, we use a modified version of genetic algorithms (similar to the one proposed in [22]) because of the poorly conditioned and extremely non-smooth meta-loss surface [74].

2.2 Sequential Decision Making

In this thesis, we are interested in a learning environment for parameterized population-based learned algorithms. In order to better explain the meta-learning framework, we introduce the terminology for Markov Decision Processes (MDPs) and, then Partially Observed Markov Decision Process (POMDPs).

2.2.1 Markov Decision Processes (MDP)

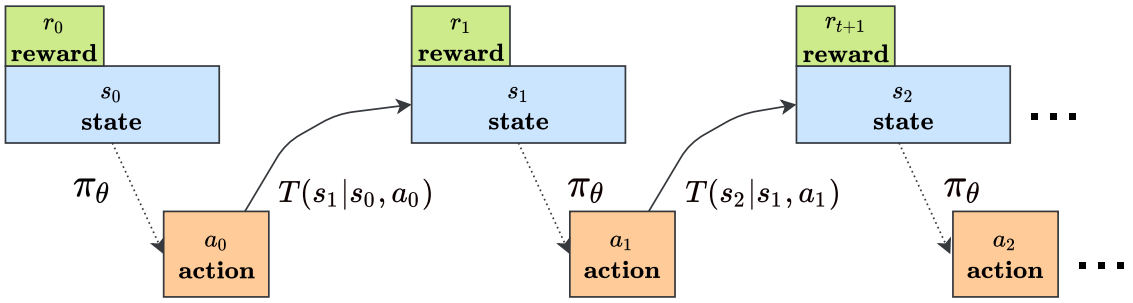

Figure 2.1: Overview of a Markov decision process (MDP) where the policy π_θ receives rewards (green) due to actions (red) based on the states (blue) of the system.

The Markov decision process (MDP) is a well-known framework that can be applied whenever we have a decision-making rule (i.e., policy) π making decisions in a system described by:

- $\mathcal{S}$: the set of system states.
- $\mathcal{A}$: the set of actions available to the agent.
- $T(s'|s, a)$: the transition model defined by the conditional transition probabilities for each state transition, i.e., when the system changes from state s to the next state s' when the agent executes an action a
- $R(s', a, s)$: a reward function that associates a feedback to each 3-tuple (state, action and next state), i.e., the reward obtained by the agent if it executes an action a when the system transitioned from the state s to next state s'.

Given those components, the dynamics of a time-discrete MDP proceed according to the following procedure: At each time t, an agent selects an action a. As a result, the system goes from the current state s to the next state s' and the agent receives a reward r. We consider a finite-horizon scenario where this procedure continues until a T max number of iterations, i.e. when $t = T$. We refer as *episode trial* the full run of an agent on the MDP and we represent it as the following sequence:

$$s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_H, a_H, r_H \tag{2.2}$$

The agent is parameterized as $\pi(a|s)$ which defines a distribution of probability over different actions. To specify the MDP completely, in addition to the model, the policy and the aforementioned procedure, we need to specify a performance criterion or objective function. We consider an objective with finite horizon H as:

$$J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, \pi_\theta(s_t) = a_t)\right] \tag{2.3}$$

where the states evolves according to the transition model $T(s_{t+1}|s_t, \pi(s_t))$, H is the horizon and $\gamma \in [0, 1)$ is the discount factor. In this work, we consider the a undiscounted setting, i.e. the discount factor $\gamma = 1$.

The final goal on the MDP is to find the optimal policy:

$$\pi_\theta^* = \arg\min_\theta J(\pi_\theta) \tag{2.4}$$

In the literature, there are many algorithms for computing π^* for any MDP. When the transition and the reward function are not known by the agent, those algorithms are usually developed within reinforcement learning community [80]: the agent π_θ needs to interact with the MDP-based environment and, consequently it might improve its performance $J(\pi_\theta)$ over training time.

2.2.2 Partially Observed Markov Decision Process (POMDP)

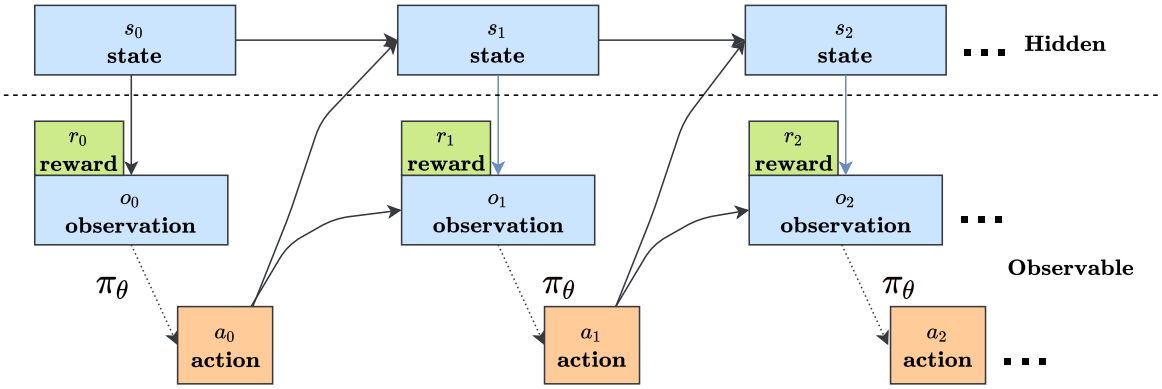

Figure 2.2: Overview of a partially observable Markov decision process (POMDP) where the policy π_θ receives rewards (green) due to actions (red) based on the observations (blue) generated by each state (blue) of the system.

In MDP, the agent has direct knowledge of the system state. However, there are many real-world challenges the agent can not directly access the state of the system. Instead, it receives an observation (possibly stochastic) that depends on the current state. In these cases, the system is modeled as a Partially Observed Markov Decision Process (POMDPs).

POMDPs are a rich framework to describe a number of planning problems that appears in situations with hidden state and stochastic actions. It has been applied to a range of applications (e.g. navigation [81], aircraft collision avoidance [82]). This paradigm extends the MDP framework with two more components:

- Ω: the set of system observations.
- $O(o|s', a)$: the observation transition model defined by the conditional transition probabilities for each state transition, i.e., the probability that the agent observes o when the system reaches state s' with the agent's action a.

A general formulation of a POMDP [83; 84] can be defined as the tuple $(S, A, \Omega, T, O, R, \gamma)$ where S, A and Ω are the set of states, actions and observations; T is a set of $T(s'|s, a)$ conditional transition probabilities for each state transition $s \to s'$ and O is the set of $O(o|s', a)$ conditional observation probabilities; Finally, the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ associates a reward to each pair of state-action and we consider a undiscounted setting, i.e. the discount factor $\gamma = 1$.

We focus on *finite-horizon* problems: given a horizon H, an agent selects an action each time step t until a terminal state s_H. Since the system is partially observable, the planning agent may not be able to observe its exact state, introducing the use of observation o_t instead of s_t. The agent receives a reward $R(s_t, a_t)$ for each state during the interaction in the particular POMDP.

We denote the agent as a policy π_θ. Similar to the MDP, the solution of a POMDP is a policy π^*_θ which maximizes the expected total reward of a full episode:

$$\pi^*_\theta = \arg\max_\theta \mathbb{E}\left[\sum_{t=0}^{H} R(s_t, a_t)\right] \tag{2.5}$$

where a_t is the output of a given policy $\pi_\theta(a_t|o_0, ..., o_t, a_0, ..., a_{t-1}, r_0, ..., r_{t-1})$.

2.3 Learning to Optimize Framework

This section presents our proposed learning framework. We refer to it as Learning-to-Optimize POMDP (LTO-POMDP) since its dynamics are similar to the Meta-RL POMDP [58]. Our framework's goal is straightforward: to meta-learn a population-based algorithm that is robust over a distribution of optimization problems, which are our problems of interest. To that end, we parameterize the optimizer (Equation 1.7) as a policy π_θ on the LTO-POMDP. Then, we train it in a meta-learning fashion. To provide the intuition behind the framework, we first consider a single run (*inner-optimization*) of the algorithm. Then, we focus on the meta-loop (*outer-optimization*) of the framework, which is responsible for training the algorithm.

Algorithm 7 A single run in Learning-to-Optimize POMDP

At time $t = 0$:
the system samples a task $w_k \sim p(w)$
the system is initialized in the initial state $s_0 = \{\emptyset, w_k\}$
the system generates an initial observation $o_0 = \{\emptyset\}$

for t = 0,1,2,...,T **do**

1. Based on the available information $\mathcal{I}_t$:

$$\mathcal{I}_0 = \{\emptyset\} \qquad\qquad \mathcal{I}_t = \{\emptyset, a_0, r_0, \ldots, o_{t-1}, a_{t-1}, r_{t-1}\}$$

the policy takes an action a_t given the current observation o_t:

$$a_t \sim \pi_\theta(a|o_t, \mathcal{I}_t) \in \mathcal{A}$$

2. The policy receives a reward $r_t = r(s_t, a_t)$ for choosing action a_t

3. The current state s_t evolves based on the task w_k to the next state s_{t+1}:

$$s_{t+1} = \{o_{t+1} = w_k(a_t), w_k\}$$

4. The system shows an observation $o_{t+1} \in \Omega$ corresponding to the state s_{t+1}:

$$o_{t+1} = \{w_k(a_t)\}$$

5. The policy updates its available information as:

$$\mathcal{I}_{t+1} = I_t \cup \{a_t, r_t, o_{t+1}\}$$

end for
The policy receives a final reward $r_T = r(s_T, a_T)$ and the process terminates.

Inner-optimization We describe a single run of π_θ in Algorithm 7. Let $\mathcal{W}$ be a space of tasks and $p(w)$ be a distribution over tasks $w \in \mathcal{W}$. At time $t = 0$, the run begins by sampling a task $w_k \sim p(w)$ that remains the same from the start to the end of the episode. We define the current state s_t as the set $\{o_t, w_k\}$ where o_t is the current observation. In other words, this particular POMDP defines an episode-fixed task $w_k \sim p(w)$ as the only unobserved state.

Each component of the LTO-POMDP (i.e., the states, observations, actions and rewards) is based on a standard population-based search. The current task w_k defines the dynamics of what the optimizer observes during the optimization rollout. Specifically, we consider a hidden state $s_t = \{P_Y^{t-1}, w_k\}$ that generates deterministically the observation $o_t = P_Y^{t-1}$ as the population evaluation of a given action $a_t = P_X^t$, which are the search points at iteration t. In the context of black-box optimization, it means that the algorithm does not have access to any information about the task w_k that it is currently optimizing and it can only observe the evaluations o_t. Then, the algorithm π_θ selects the search points to be evaluated based on all available information: $\mathcal{I}_t = \{\emptyset, P_X^0, r_0, P_Y^0, P_X^1, r_1, P_Y^1, \ldots, P_Y^{t-1}, P_X^{t-1}, r_{t-1}\}$.

Finally, we consider that the execution of the algorithm parameterized with θ (Equation 1.7; Algorithm 1) can be viewed as the execution of a policy in the LTO-POMDP. Therefore, the policy π_θ is equivalent to the Equation 1.7, and it can be implemented as various population-based search algorithms as we have seen in Chapter 1.

Outer-optimization Besides the dynamics of a single run (i.e., the inner-optimization), we need to define the *outer-optimization* loop. We consider that the algorithm does not have access to the system's transition model. Hence, the policy needs to interact with LTO-POMDP to improve its performance over time. The parameters θ of a policy π_θ are trained by optimizing the overall performance across tasks sampled from $p(w)$. We need to define a way to evaluate the performance of a given policy π_θ for a given distribution of tasks $p(w)$, i.e. a *meta-loss function*. More concretely, we define our meta-objective as follows:

$$\min_\theta \mathop{\mathbb{E}}_{w_k \sim p(w)} \left[J^k(\pi_\theta) \right] \tag{2.6}$$

where $J^k(\pi_\theta)$ is the performance of the policy π_θ for the task $w_k \sim p(w)$.

An overview of the framework is illustrated in Figure 2.3. In practice, the framework may be flexible based on how to define the distribution of optimization tasks (blue) and how to aggregate the rewards of many runs to evaluate the performance $J(.)$ (green). We will clarify them in the following subsections.

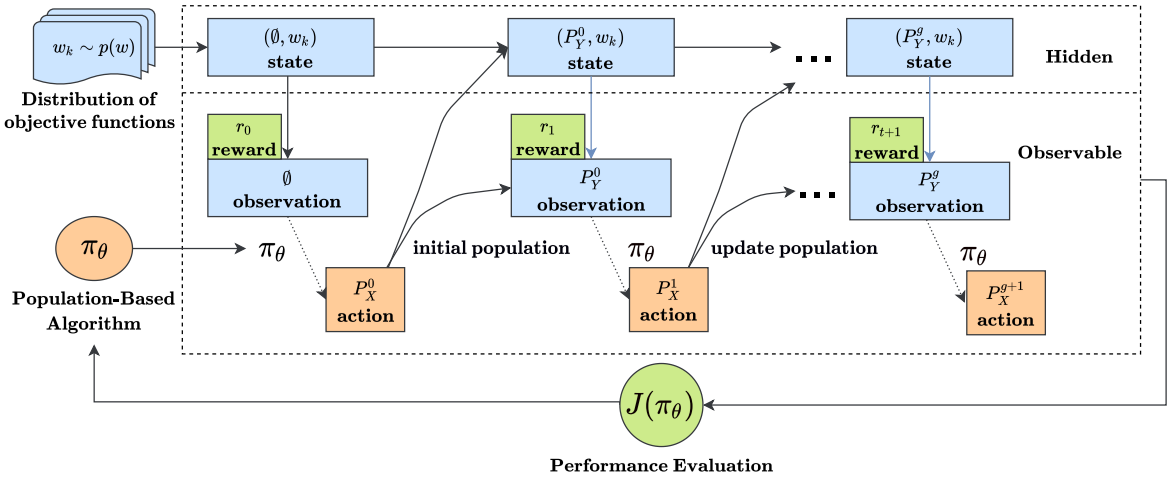

Figure 2.3: Overview of the learning-to-optimize POMDP (LTO-POMDP). The optimizer π_θ (red) is updated after the evaluation by the meta-loss function $J(\pi_\theta)$ (green) over a distribution of tasks (blue) in the LTO-POMDP.

2.4 Performance Measurement

Notice that, in our context, π_θ represents a parameterized (possibly stochastic) learned optimizer. When we compare different optimizers, we should be aware that some optimizers may have a small probability of solving the task and solve it fast, whereas others may have a higher probability of success but be slower. Therefore, to obtain a robust evaluation of an optimizer, we consider a commonly used approach in benchmarking stochastic optimizers: the conceptual *restart algorithm*. We measure the performance of π_θ as the expected number of function evaluations (FE) to reach a certain function value (*success criterion*) of w_k by conducting independent restarts of π_θ.

Let $p_s \in (0, 1]$ be the probability of success of π_θ to reach a certain function value of w_k and $\mathrm{FE}_{\max}$ be the maximum number of function evaluations, i.e., the number of evaluations done when π_θ does an unsuccessful run. Then, the performance $J(\pi_\theta, w_k) = \mathrm{FE}(\pi_\theta, w_k) = \mathrm{FE}_{\theta,w_k}$ is a random variable measuring the number of function evaluations until a *success criterion* is met by independent restarts of π_θ on the task $w_k \sim p(w)$:

$$\mathrm{FE}_{\theta,w_k} = N * (\mathrm{FE}_{\max}) + \mathrm{FE}^{\mathrm{succ}}_{\theta,w_k} \tag{2.7}$$

where N is the random variable that measures the number of unsuccessful runs of π_θ required to reach once the success criterion and $\mathrm{FE}^{\mathrm{succ}}_{\theta,w_k}$ as the number of evaluations for the successful run of π_θ to reach the criterion.

Now we use the fact that $N \sim \text{BN}(r = 1, p = 1 - p_s)$ follows a negative binomial distribution and the expectation of this distribution is $\frac{rp}{1-p} = \frac{1(1-p_s)}{1-(1-p_s)} = \frac{1-p_s}{p_s}$. Then, We take the expectation $\mathbb{E}[\text{FE}_{\theta,w_k}]$ of FE_{θ,w_k}:

$$\mathbb{E}[\text{FE}_{\theta,w_k}] = (\mathbb{E}[N])\text{FE}_{\max} + \mathbb{E}[\text{FE}^{\text{succ}}_{\theta,w_k}] = \left(\frac{1-p_s}{p_s}\right)\text{FE}_{\max} + \mathbb{E}[\text{FE}^{\text{succ}}_{\theta,w_k}]. \tag{2.8}$$

Estimation of the performance measurement

In practice, we do not know the value p_s and $\mathbb{E}[\text{FE}^{\text{succ}}_{\theta,w_k}]$ and we need to estimate each of those terms.

We can estimate the probability of success p_s as:

$$\hat{p_s} = \frac{\#\text{ successful runs}}{\#\text{ runs}} \tag{2.9}$$

This estimator is the maximum likelihood estimator for p_s and is unbiased.

We can estimate the expected number of function evaluations for successful runs $\mathbb{E}[\text{FE}^{\text{succ}}_{\theta,w_k}]$ as:

$$\widehat{\mathbb{E}[\text{FE}^{\text{succ}}_{\theta,w_k}]} = \frac{\#\text{ total evaluations for successful runs}}{\#\text{ successful runs}} \tag{2.10}$$

Taking the previous meta-objective (Equation 2.6), the expected number of evaluations (Equation 2.8) and its estimators (Equations 2.9, 2.10), we can write the final meta-objective as:

$$\min_{\theta} \mathop{\mathbb{E}}_{w_k \sim p(w)} [J(\pi_\theta, w_k)] = \min_{\theta} \mathop{\mathbb{E}}_{w_k \sim p(w)} [\text{FE}_{\theta,w_k}] \tag{2.11}$$

$$= \min_{\theta} \mathop{\mathbb{E}}_{w_k \sim p(w)} \left[\left(\frac{1-\hat{p_s}}{\hat{p_s}}\right)\text{FE}_{\max} + \widehat{\mathbb{E}[\text{FE}^{\text{succ}}_{\theta,w_k}]}\right] \tag{2.12}$$

Our experiments show that this encourages the learned policy to be sample efficient (when it can solve different tasks in a few steps) and robust to a premature convergence (when it can solve more tasks in more steps). However, other papers have used more common reward functions [5; 85]. We believe that different meta-loss functions can generate various learned population-based optimizers that exploit different classes of problems.

2.4.1 Task Distribution

The last component of the proposed meta-learning framework relates to the concept of tasks and their distribution. The task distribution in our context can describe any class of optimization problems including hyperparameter optimization [75], black-box optimization [76]

or reinforcement learning problems [86]. We will consider only continuous optimization problems for the experiments in the remainder of the paper, given their high applicability to many real-world tasks and comparability through benchmarking with other population-based approaches.

Let f_ν be an objective function and ν_i the configurations of an instance of the function indexed by $i = 1, 2....$ In order to be consistent with the definition of a task in the meta-learning literature, let $\mathcal{F}$ be the space of objective functions and $\mathcal{V}$ be the space of configurations of objective functions. We consider that a task instance is drawn from a distribution $p(\mathcal{C})$ over a class of optimization problems, such that $w_k \sim p(\mathcal{C})$, where $\mathcal{C} = \mathcal{F} \times \mathcal{V}$.

In practice, we often do not have access to this distribution, and drawing a task may consist of picking a specific (parameterized) instance of a given function family (e.g., a quadratic function, accuracy of SVM) from a set of available functions. The configuration ν_i can represent coefficients as well as different transformations of the function f_ν. We consider different instances to correspond to variations of the same problem. For example, in synthetic functions, optimal values can be shifted or applied rotation and translation on the search space. In hyperparameter tuning, we can see different instances as different losses that vary based on different datasets or training regimes. The notion of the objective function f_ν and its configuration ν_i is thus flexible and defines a problem class. Finally, we represent a task w as a tuple (f_ν, ν_i).

2.5 Conclusion

In this chapter, we reviewed different approaches for learning to optimize. We briefly described the representation, task distribution, and meta-optimizer used in this work as well as some background knowledge in MDPs and POMDPs. Then, we explain our POMDP-based framework in detail, which is used to train and evaluate population-based optimizers. We also formally define our meta-objective and the task distributions. In the next chapter, we introduce the policy architecture and the meta-optimization procedure that we chose to learn population-based algorithms. Then, we conclude with the experimental results.

Chapter 3

Experimental Results and Discussion

In the previous section, we have detailed each component of a meta-learning framework for population-based algorithms: the *inner* and *outer* optimization run in a POMDP-view, how the policy is evaluated, and how the distribution of tasks is defined in the context of a black-box optimization. In this section, we use those components to address the hypothesis that the learning-to-optimize POMDP can yield an efficient optimizer in different scenarios. Besides, we also validate the learned optimizer's generalization power when defining a broader class of problems.

This chapter is organized as follows. First, the experimental setup is presented with all implementation details of the policy architecture, the *outer*-optimizer used in the experiments and the evaluation procedure. In other words, how we represent the learned algorithm, how we meta-train it and how we can compare it with other hand-engineered optimizers. Later, we show the experimental analyzis of the results in two scenarios: 1) the optimizer should exploit a structure of a particular benchmarking black-box function to succeed; 2) the optimizer faces a broader class of problems, and it should exploit them while not suffering from premature convergence. The implementation of the framework and all scenarios is available at `https://github.com/optimization-toolbox`.

3.1 Policy Architecture

The policy architecture π_θ is the representation of the learned population-based optimizer. It represents the decision-making rule of the algorithm, which defines a mapping function from the y-values P_Y^{t-1} of the previous population to the current chosen search points P_X^t (see Figure 2.3). For example, in the case of a simple Random Search, the policy ignores the state P_Y^{t-1} and sample a batch of search points from a uniform distribution over the domain; and choose the set P_X^t as an action at time t in the framework. In the case of CMA-ES, the policy has access to a covariance matrix, which is updated based on the current state P_Y^{t-1} such that the likelihood of previously successful candidate solutions is maximized [34].

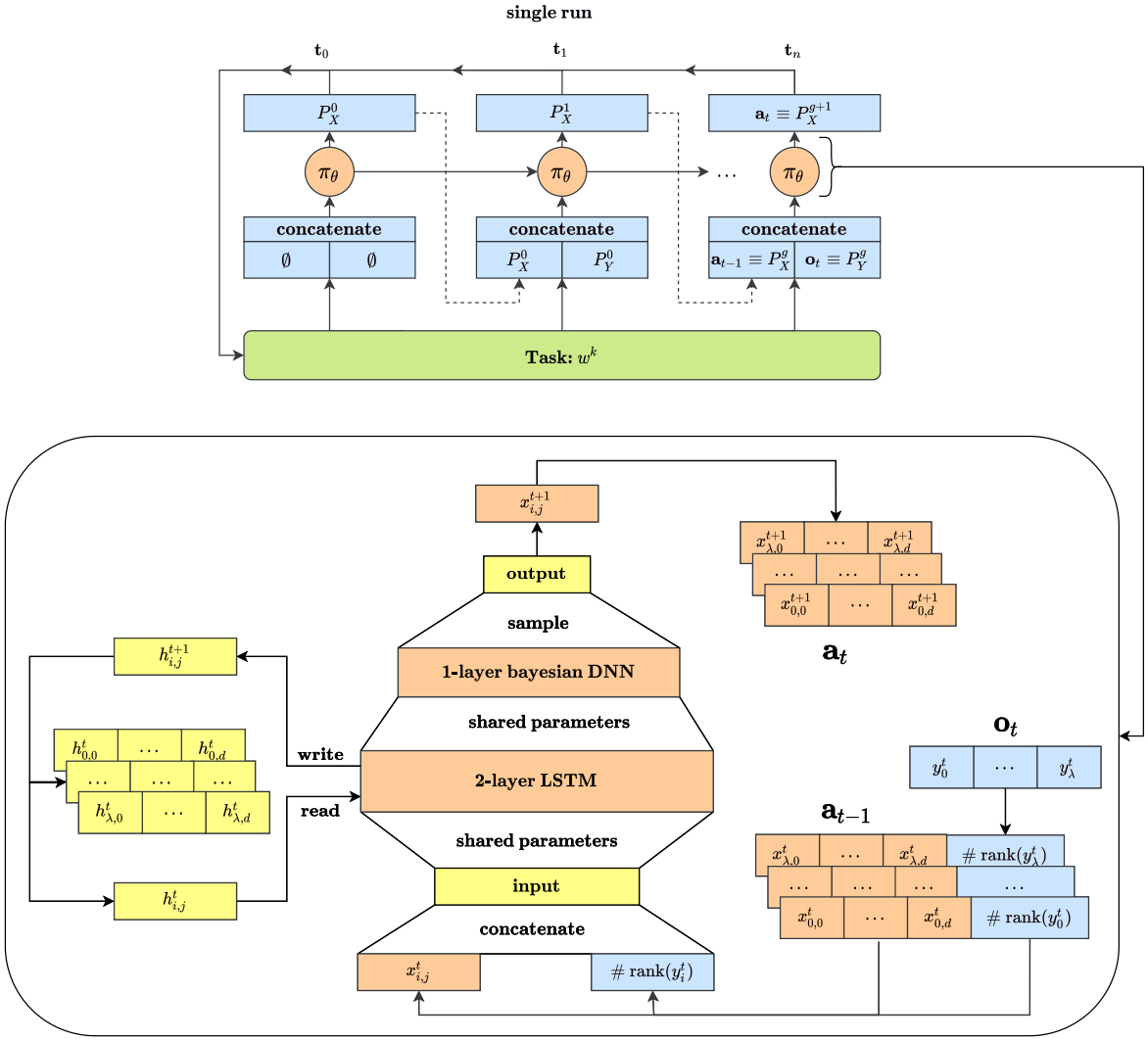

Figure 3.1: The inner-optimization example (left): The algorithm π_θ receives the population P_X^t and the evaluation P_Y^t in order to optimize a task w_k. Learned Population-Based Optimizer (LPBO) (right): The policy architecture π_θ using 2-layers LSTMs followed by a 1-layer Bayesian Neural Network. The LSTM module outputs a hidden memory and the next population for each (parameter, individual) pair in the course of the optimization.

In our case, we use a function approximator (i.e., a deep neural network) that parameterizes possible hand-engineered search mechanisms, and we believe this can automatize a better algorithm for an user-specific scenario. In all experiments, the trained policies use the same architecture (including hyper-parameters). We use a stacked 2-layers LSTMs with 64 neurons in the hidden layer followed by a 1-layer Bayesian Neural Network. We use the LSTM to model the dependency on all previous populations (common choice in POMDP-based meta-reinforcement tasks [87; 58]), and we use the Bayesian Neural Network to represent stochastic policies since most population-based algorithms have stochastic search mechanisms. We unified the domain search of all tasks to be [-1, 1]. Therefore, we use tanh as an activation function for the output. We initialize all parameters using a normal distribution ($\sigma = 0.5$).

We consider two other particularities of the policy. First, **scalability** to the order of the dimensions of a given problem. Based on recent works on learnable optimizers [63; 66], we use a *coordinate-wise policy* in order to be able to optimize in higher dimensions. It means that the policy has different behaviors for each individual and dimension based on its memory. We use different memories (hidden states) in practice, and we make $\lambda * D$ predictions at each population update.

Second, **invariance** under increasing transformation of the current problem. A major design option for many optimization algorithms is related to invariance: the ability of a method to generalize from a single problem to a class of problems. An important invariance principle is a monotonically increasing transformations of f (i.e. the performance of the optimizer in f is the same in $f, f^3, f \times 2 * |f|^{\frac{-5}{9}}, \ldots$). Therefore, we replace all the evaluations on f with an adaptive transformation of f to represent how the observed values are relative to other observations in the current step [88]. In practice, we use the ranking of the evaluations instead of its absolute value. Those two choices allow us to reduce the number of parameters of the policy to be optimized since the input is only two numbers and the output a single one (Figure 3.1), while it provides additional features related to *scalability* and *invariance*.

3.2 Meta-Optimizer

The Learning-to-Optimize POMDP framework reduces the problem of learning a population-based algorithm to finding an optimal policy on a POMDP, which can be solved using different approaches. Although we could have used reinforcement learning methods to solve it [5], we choose an evolutionary algorithm.

They are a natural choice to optimize a non-differentiable meta-loss function (Equation 2.12) due to its black-box optimization nature. Recent works show that evolutionary algorithms are competitive in reinforcement learning environments [21]; they are more robust to extremely non-smooth landscapes [73] and have high parallelization power [88]. In this way, we see the policy agent as a flat vector of numbers (the parameters of the neural network) that is

associated with its evaluation (Equation 2.12), and we want to find the best setting for those parameters. Mathematically, we want to optimize a black-box meta-loss function $J(\theta)$ with respect to the input vector θ (the parameters/weights of the network).

Thus, we use a slightly modified version of DeepGA presented in [22]. Specifically, this is a genetic algorithm which maintains a population of parameters $(\theta_1, \ldots, \theta_\lambda)$ represented by a list of seeds. The seeds define the mutations applied to each individual through generations. Thus, each CPU core can evaluate a population of policies separately. Instead of sending thousands of thousands of parameters to each other, they can communicate using a list of seeds corresponding to a parameter θ_i. The fitness evaluation is the performance measurement explained in section 2.4 applied to a batch of functions. We use 512 as the population size, 5 as the number of elite individuals, 20 as the number of parents in each generation, and $\sigma_0 = 0.3$. The difference from the original paper is that they use a fixed σ in all generations while we use a decaying strategy where we update $\sigma_t = \beta * 0.95$, with $\beta = 0.95$ and a minimum $sigma_{\min} = 0.01$ over 100 generations.

3.3 Rationales on the Evaluation Procedure

The last step before we start the experimental analysis is to define a way to evaluate the learned optimizer. To this end, we use a general black-box optimization benchmarking: The COCO platform [76]. This platform's original purpose is to automatize black-box optimization benchmarking procedure providing data logging facilities, data post-processing that produces various plots and tables, and other solvers' empirical results that can be used for comparison. However, in our context, we want to extend it to a learning problem perspective.

In this work, the COCO interface was wrapped in the Learning-To-Optimize POMDP implementation using the OpenAI Gym [89], which is a tool for reinforcement learning tasks. We provide an interface to generate different functions, and we can aggregate them in a class of problems. The advantage of using such an approach is that we can split a set of functions to be the train, validation, and test dataset like any other machine learning task.

We have generated one thousand different instances of each function in all experiments, and we use 100 as training, 100 as validation, and 800 as testing. In other words, the policy is trained/validated/tested in $100 * N$ different functions selected. We refer the reader to check each function's details in the official COCO platform documentation: `https://github.com/numbbo/coco`.

This thesis lies in between two mathematical fields: meta-learning, which aims to evaluate the fast-adaptation and generalization of a learnable algorithm for a specific learning scenario; and black-box optimization, which aims to assess the performance of a given optimizer in some problems of interest. Therefore, we have two methodologies to evaluate the learned optimizer.

From a learning perspective, we show a meta-loss plot for train and validation tasks (Example in Figure 3.3), which is the standard evaluation of a machine learning algorithm. This plot measures the performance of the learned algorithm in a train set and a validation set to display the algorithm's generalization. The y-axis represents the meta-loss function described in Section 2.4. The x-axis presents the best individual of the meta-optimizer population in the current step. In the case of the training loss, the plot will always be a decreasing function because we have selected elite individuals > 1 (i.e., the best individual is always kept each generation, then the best evaluation never decreases). In the validation loss case, we plot the best individual's performance in the population for the validation set. Notice that each population's best individual can change in generations, and the evaluation value might change. Finally, we expect both losses with similar convergence values.

From an optimization perspective, we want to know the learned algorithm's performance on an inner-optimization loop. Therefore, we use the same benchmarking procedure on the COCO platform. We aggregate the results using the test dataset, which the learned optimizer did not have access during its meta-training. The idea behind aggregation is to compute a statistical summary over a set of problems of interest over which we assume a uniform distribution. From a practical perspective, we believe in facing each problem with similar probability and have no simple way to distinguish between them to select a solver accordingly. We use the same strategy proposed in the COCO platform: the Empirical cumulative distribution functions of runtimes (runtime ECDFs), also denoted as (empirical) runtime distributions (See Definition 3.3.2). Empirical distribution functions over runtimes of optimization algorithms are also known as data profiles [90].

Definition 3.3.1 (Cumulative distribution function - CDF)**.** The Cumulative distribution function (CDF) of a random variable X is defined as:

$$F(x) = P(X \leq x), \text{for all } x \in \mathbb{R}.$$

Definition 3.3.2 (Empirical cumulative distribution function of runtimes: runtime ECDFs)**.** Let $(\mathrm{FE}_1, \ldots, \mathrm{FE}_n)$ be independent, identically distributed real random variables, which represent the runtime (i.e., number of function evaluations) of an optimizer with a particular cumulative distribution function F(t). Then, the runtime ECDFs $\widehat{F}(t)$ is defined as:

$$\widehat{F}(t) = \frac{\#\text{ elements in the sample} \leq t}{n} = \frac{1}{n} \sum_{n}^{i=1} \mathbb{1}_{\mathrm{FE}_i \leq t}, \text{where } \mathbb{1}_A \text{ is the indicator of event A}$$

We use an estimation of the cumulative distribution function that generated the runtimes in the optimizer runnings to define the runtime ECDFs. Notice that the empirical distribution function can be read as the y-axis being an independent variable: It means that for any fraction of problems (y-value), we see the maximal runtime observed on these problems on the x-axis. For example, in Figure 3.3, we see that the learned algorithm takes a few evaluations to solve 100% of the problems.

Besides each evaluation procedure, we also provide the baselines comparison. We have used two optimization algorithms from the literature as a policy on the Learning-To-Optimize POMDP:

- **Random Search (RS)**: we define an uniform distribution over the domain of each problem and we sample a set of search points each generation to be evaluated on the framework. Every iteration of this algorithm is not dependent on the prior iterations.
- **Covariance matrix adaptation evolution strategy (CMA-ES)** [34]: This stochastic algorithm is often stated as the state-of-the-art method for real-parameter (continuous domain) optimization of non-linear and non-convex functions. We use the implementation from https://github.com/CMA-ES/pycma. We pick a random point for the interval [-0.8, 0.8] and $\sigma = 0.3$, which are the default values for COCO benchmarking.
- **Learned Population-Based Optimizer (LPBO)**: Our trained algorithm described in the sections 3.1 and 3.2. We parameterize the population-based solver using deep neural networks while it is trained using a modified version of genetic algorithm in the Learning-To-Optimize POMDP.

3.4 Learning Population-Based Algorithms Implementation

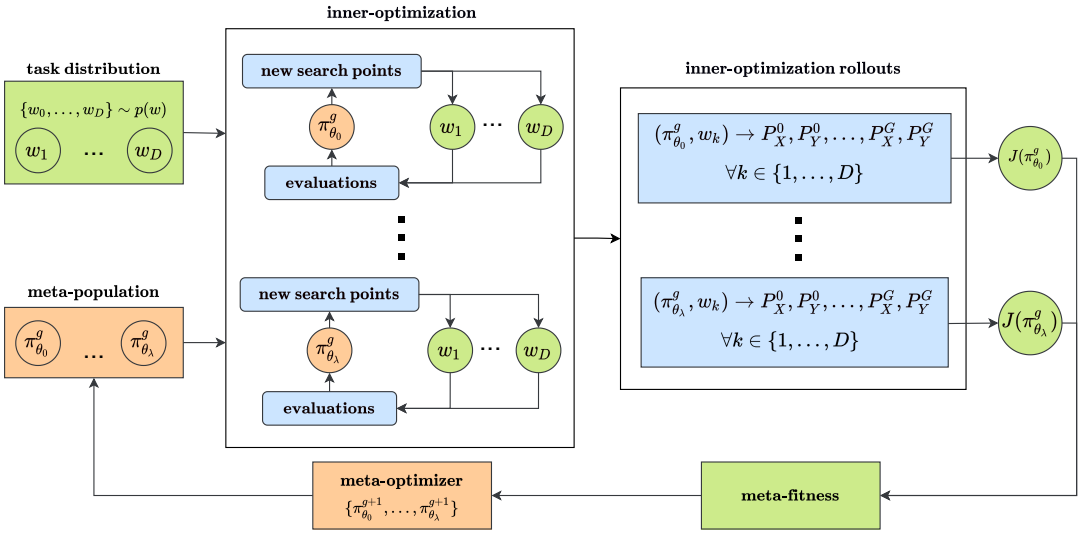

Figure 3.2: Overview of Meta-Learning Population-Based Algorithms Approach: The meta-optimizer (red) starts with a random population of optimizers. Then, they interact with a set of BBO tasks (yellow) and generate an optimization rollout (blue). They are evaluate following the meta-fitness function $J(.)$ (green), and a new set of optimizer is generated. The loop continues until a stopping criteria is reached.

In the previous sections, we have explained our POMDP-based meta-learning framework, the meta-optimizer, the policy architecture, and how we evaluate it. All those components are jointly defined to train population-based algorithms that fast-adapt for a given black-box optimization context. This section provides an overview of the approach to learning population-based algorithms developed so far in this thesis.

As previously mentioned, meta-learning can be viewed as a *bi-level* optimization. We use the prefix *inner* for the components in the first level of the optimization, where the optimizer π_θ with parameters θ is being used to optimize the decision variables of a given task w. Also, the prefix *meta*, for the components in the second level of optimization, where the meta-optimizer is being used to optimize the parameters θ of the learned optimizer π_θ. The approach is depicted in Figure 3.2. Since the chosen meta-optimizer is DeepGA, we start by initializing a population of learned algorithms $\{\pi^{t=0}_{\theta_0}, \ldots, \pi^{t=0}_{\theta_\lambda}\}$ with random parameters $\{\theta^{t=0}_0, \ldots, \theta^{t=0}_\lambda\}$. Then, we select a set of tasks (i.e., functions) $\{w_0, \ldots, w_D\} \sim p(w)$ to evaluate each of them. The evaluation consists in generating all inner-optimization rollouts $P^0_X, P^0_Y, \ldots, P^H_X, P^H_Y$ for each task. Then, we can use Equation 2.12 to estimate the meta-fitness $J(.)$ for each learned algorithm. Each inner-optimization loop can be parallelized regarding each individual π_θ or even for each pair (π_θ, w_k). This is important when the evaluation of a single run is costly or the size of the meta-optimizer population is too high. Given that all meta-fitness values

$\{J(\pi^t_{\theta_0}), \ldots, J(\pi^t_{\theta_\lambda})\}$ and all search points $\{\pi^t_{\theta_0}, \ldots, \pi^t_{\theta_\lambda}\}$ are available, the meta-optimizer proceeds to generate the next meta-population $\{\pi^{t+1}_{\theta_0}, \ldots, \pi^{t+1}_{\theta_\lambda}\}$. And the loop continues until a stopping criteria of the meta-optimizer. Hopefully, after some generations, the best optimizer found so far π^*_θ is the one that minimizes the meta-objective function:

$$\pi^*_\theta = \arg\min_\theta \mathop{\mathbb{E}}_{w_k \sim p(w)} \left[\left(\frac{1 - \hat{p_s}}{\hat{p_s}} \right) \mathrm{FE}_{\max} + \mathbb{E}[\widehat{\mathrm{FE}^{\mathrm{succ}}_{\theta, w_k}}] \right] \tag{3.1}$$

In conclusion, we can enlighten the approach to learning a population-based algorithm when we put part of the mathematical tractability of the problem into perspective. Figure 3.2 simplifies how we see the dynamics of the learning-to-optimize POMDP (see Figure 2.3), which facilitates a more practical implementation of the proposed approach.

3.5 Experimental Analysis I: Single BBO Function

Our experiments' goals are two-fold: to study the learned optimizer's search behavior and to evaluate the benefits of our meta-learning method on broad classes of optimization problems. This section presents the results and discusses the study of the learned optimizer using the COCO platform's functions. Each function has an *exploiting strategy* that the algorithm should learn from scratch. Also, we focus on costly black-box optimization, where we have a low budget to evaluate the function. We use $10 \times D$, where D is the dimension of a given problem. We use the dimensions $2, 5$ and 10 from COCO dataset.

Linear Slope

The first scenario examined is a purely linear function that the algorithm should be capable of going outside the initial convex hull of solutions right into the domain boundary. Although the Linear Slope function can be seen as a simple function to be optimized, we have to remember that the algorithm is learning from scratch and does not have any prior knowledge of the function's structure. From a learning perspective, the optimal policy should never evaluate points outside and inside the domain boundary. We can observe in Figure 3.3, the learned algorithm can solve more problems than the baselines, and it can also solve it faster because it exploits the fact that the optimal points of the function are within the boundary. Thus, even the initial population does not happen to be outside the boundary. In contrast, CMA-ES has a slow step size increased accordingly to shift each generation's population in the boundary. Because of its random initial population, it is much slower than the LPBO, and it is required more evaluations to solve more tasks.

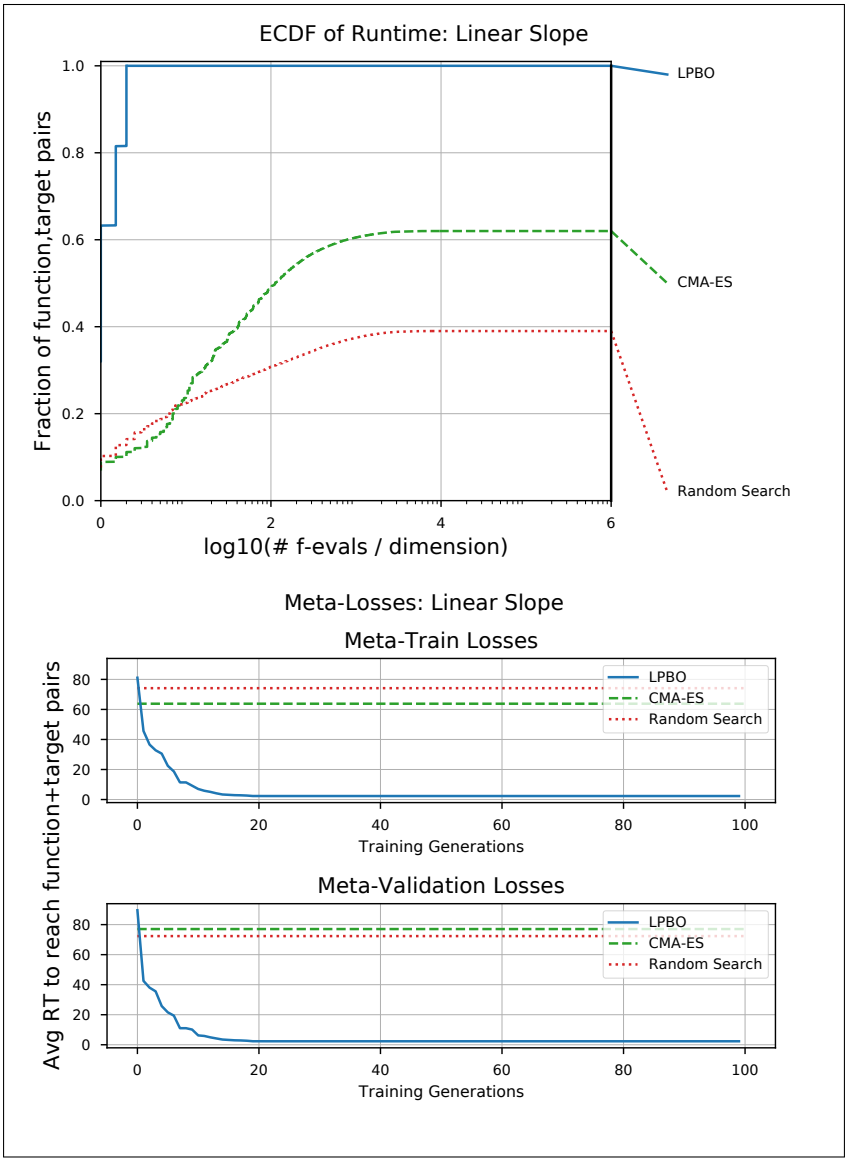

Figure 3.3: Linear Slope results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

Lunacek bi-Rastrigin Function

Another relevant result for LPBO is displayed in Figure 3.4: the Lunacek bi-Rastrigin Function. The most important insight from this function is to examine if the search behavior can be local on a global scale while being global on a local scale. When we compare the LPBO performance with other methods, we can observe that the inductive prior learned in the outer-loop achieves an efficient search on a local scale while preserving a good awareness globally. The learned algorithm is capable of solving all tasks while being much more sample efficient. The

RS and CMA-ES have similar performance here since CMA-ES would need more evaluations to adapt its covariance matrix and converge it to a local search near the optima.

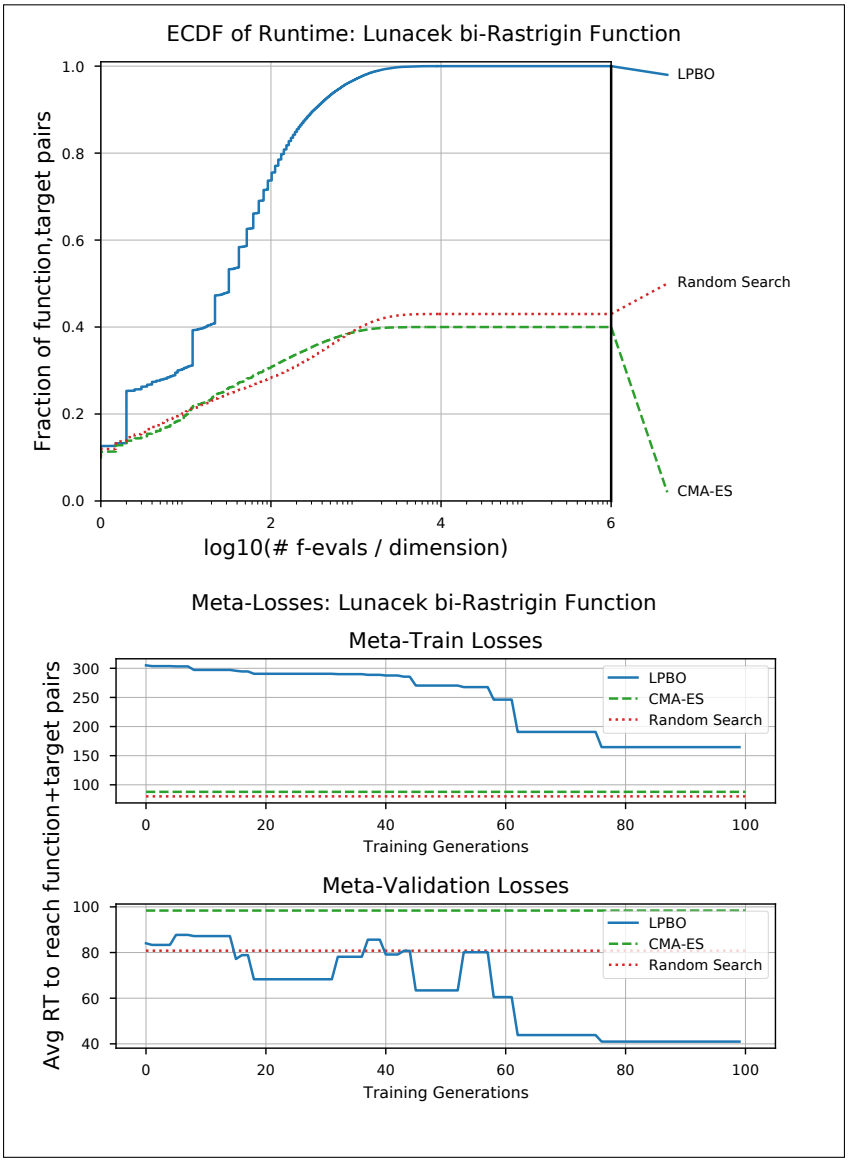

Figure 3.4: Lunacek bi-Rastrigin Function results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

Separable Ellipsoidal Function vs Nonseparable Ellipsoidal Function

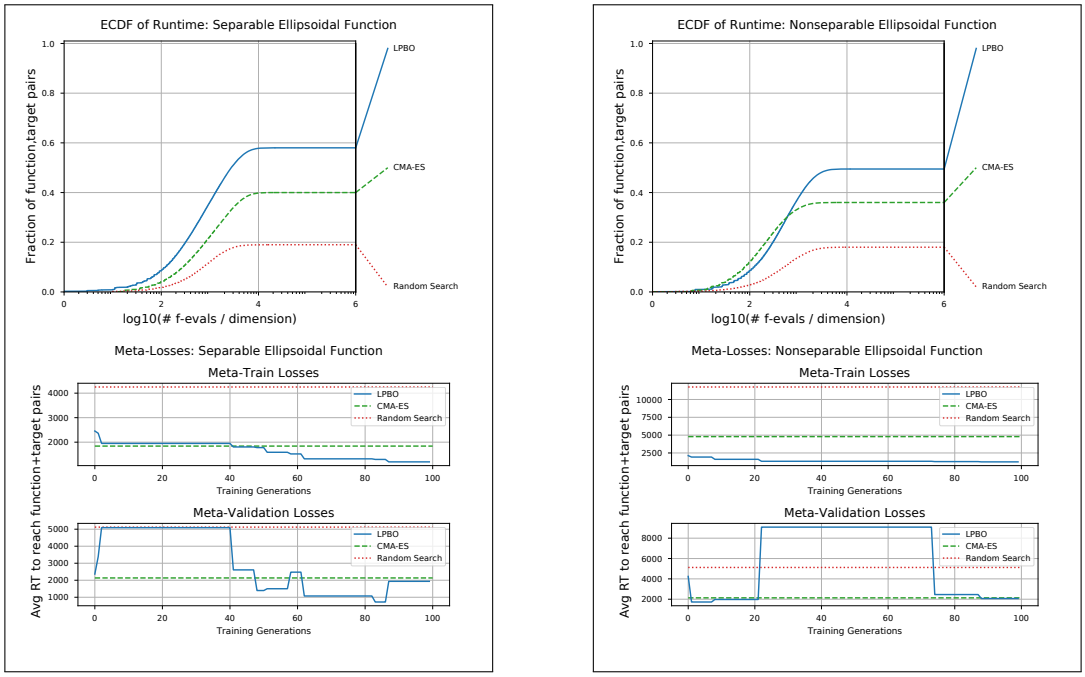

Figure 3.5: Separable Ellipsoidal Function (left) vs Nonseparable Ellipsoidal Function (right) results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

Separability is a common function property in the black-box optimization literature. In general, the search process can be reduced to D one-dimensional search procedures when the optimizer is dealing with a separable function. Consequently, we must consider the effect of a non-separability scenario for the learned algorithm. In this case, we propose to compare the performance of the Separable and the Nonseparable version of the Ellipsoidal Function in Figure 3.5. Those are also conditioned around 10^6, which implies that a small change in the input will lead to a large change to the output value.

Remember that the policy's input is only the ranking of the evaluations instead of its absolute value. Therefore, the decision-making rule relies solely on how the individuals are evaluated *relative* to each other. Considering that the policy uses the same **invariance** principle of the CMA-ES method [88], we expect to have similar performance for both scenarios. Although the LPBO can not solve all tasks, it can achieve a better performance than CMA-ES and RS.

Step Ellipsoidal Function

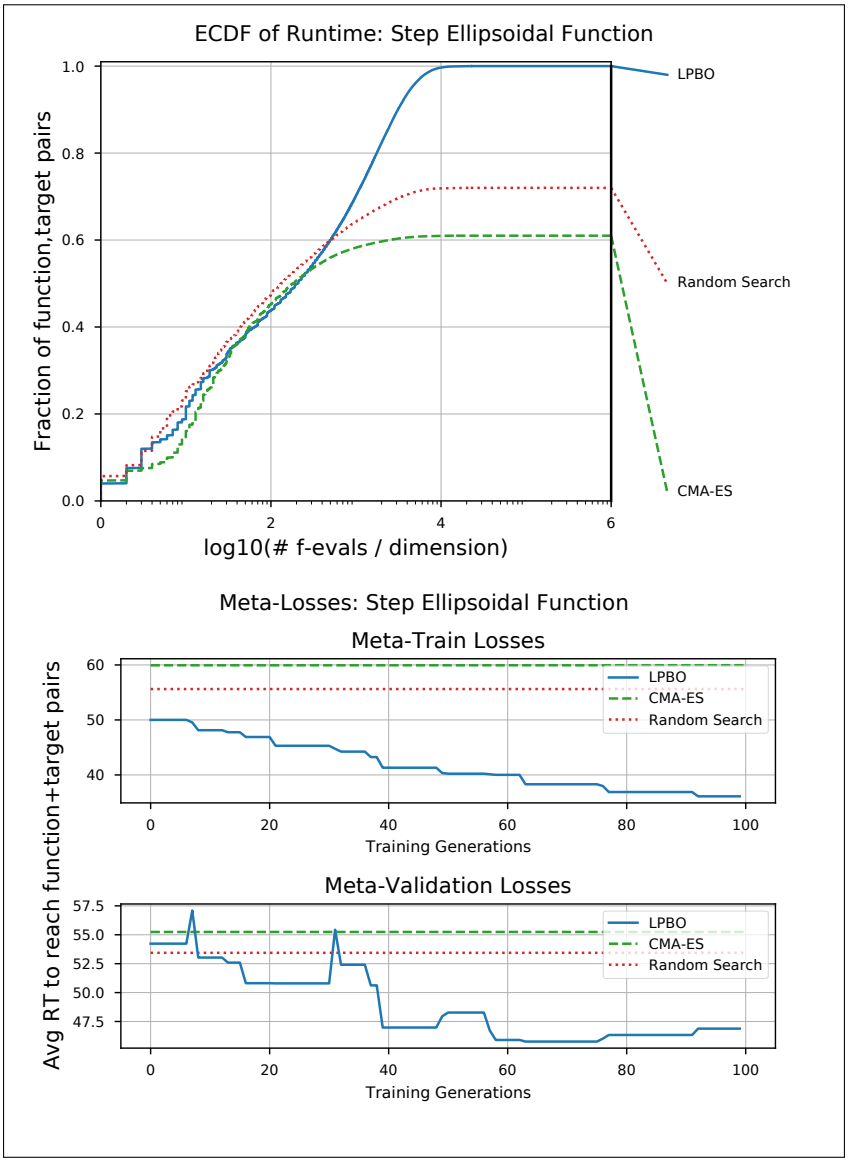

Figure 3.6: Step Ellipsoidal Function results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

The Step Ellipsoidal Function is unimodal, nonseparable, with conditioning about 100. Also, the main characteristic of this function is that it consists of many plateaus of different sizes. Consequently, those plateaus make this function's landscape with zero gradients almost everywhere, except for a small area close to the global optimum. The results are displayed in Figure 3.6. We observe that the policy's stochastic weights might explore the landscape efficiently compared to the CMA-ES. The CMA-ES is limited to adapt a Normal Distribution, but the

expressiveness of bayesian neural networks can provide better flexibility in representing the adapted distributions. In this particular case, RS is more efficient than CMA-ES because it can have a better exploration strategy in a low number of evaluations regime.

Büche Rastrigin Function

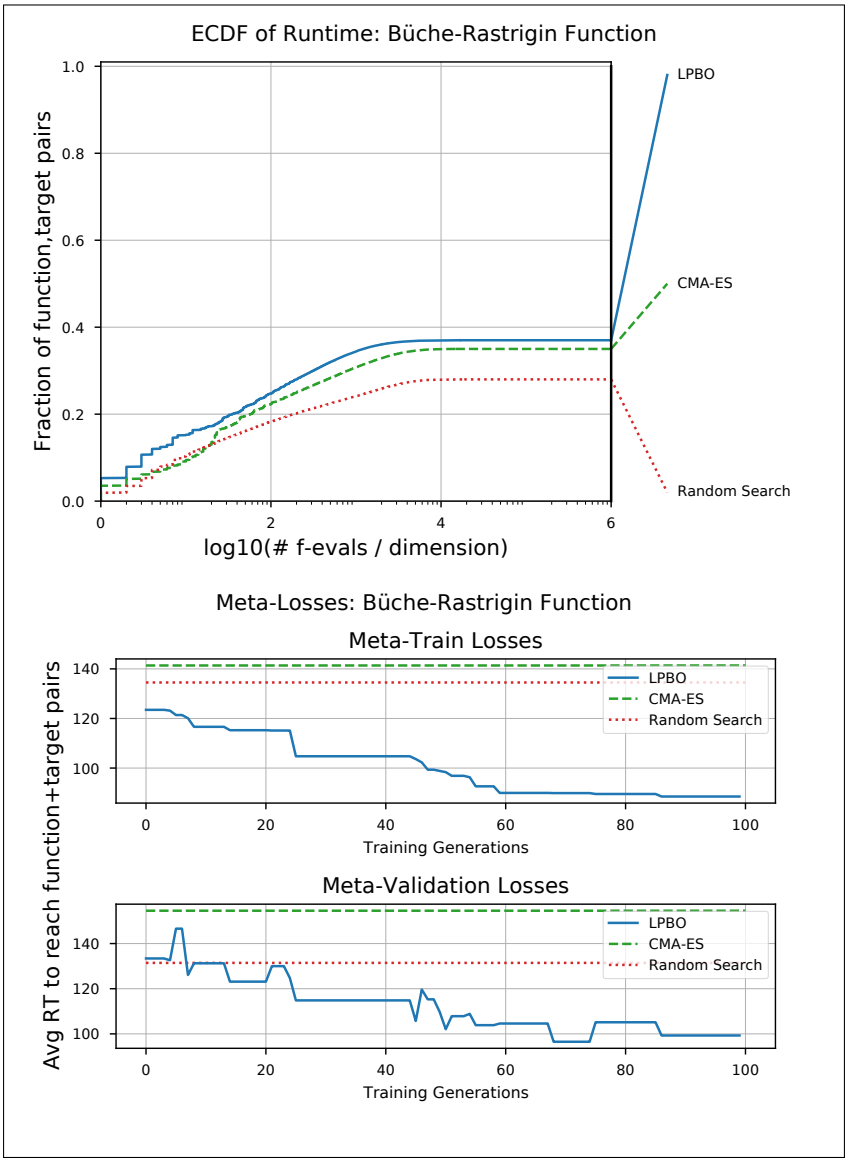

Figure 3.7: Büche Rastrigin Function results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

The Büche Rastrigin Function is a highly asymmetry function. Stochastic search procedures often rely on Gaussian distributions to generate new solutions, and it has been argued that

symmetric benchmark functions could favor these operators. Thus, we need to investigate the performance of the learned algorithm in those scenarios.

In our low budget regime, CMA-ES has a low performance. Unexpectedly, its search mechanism based on Gaussian distributions does not appear to make a significant impact on its performance. RS does not use this mechanism and still have a low performance. The learned algorithm has slightly superior performance than CMA-ES and RS; however, it is inconclusive if it can fast-adapt the population over asymmetric functions. A further investigation comparing the long-term behavior for all methods might be required even though the learned algorithm seems to have an adequate adaptation in this case. The results are displayed in Figure 3.7.

Composite Grewank Rosenbrock Function

The Composite Grewank Rosenbrock Function is highly multimodal. In the literature [91], CMA-ES achieves superior performance on this type of functions in comparison to other state-of-the-art global search strategies without intricate parameter tuning.

Although the learned algorithm is not competitive when more evaluations are available, we can see in Figure 3.8 that it is much more sample efficient than CMA-ES and RS. It seems to be the case that a better flexibility in the sampling strategy can lead to better performance. Further research could combine both approaches and test if the hybrid approach can have a more flexible initial search distribution for exploration while a local search using the CMA-ES mechanisms.

Schwefel Function

The last function analyzed is the Schewefel Function. COCO platform tag this function as multi-modal functions with *weak global structure*. This function has the most prominent $2 * D$ optimal points located comparatively close to the corners. Therefore, the solver should fast explore all the regions of the domain before exploiting the specific area with all local optimal points.

We can see in Figure 3.9 that the LPBO exploit this structure with the initial population. In the ECDF plot (top), LPBO can solve 28% with the initial population and update the population to better regions over the generations. Finally, CMA-ES and Random Search have similar performance due to less flexibility of the covariance matrix update related to the function's landscape.

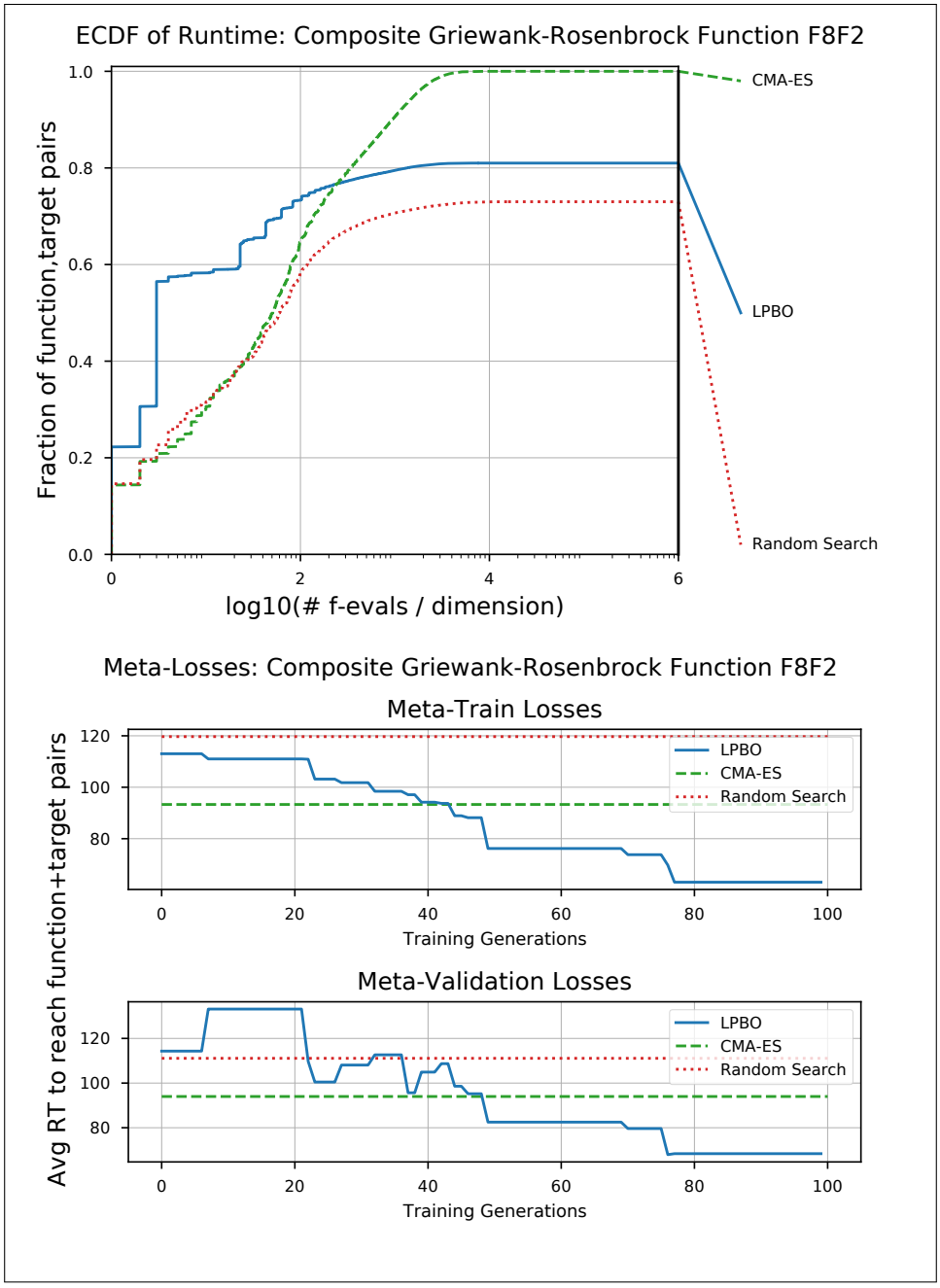

Figure 3.8: Composite Grewank Rosenbrock Function results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

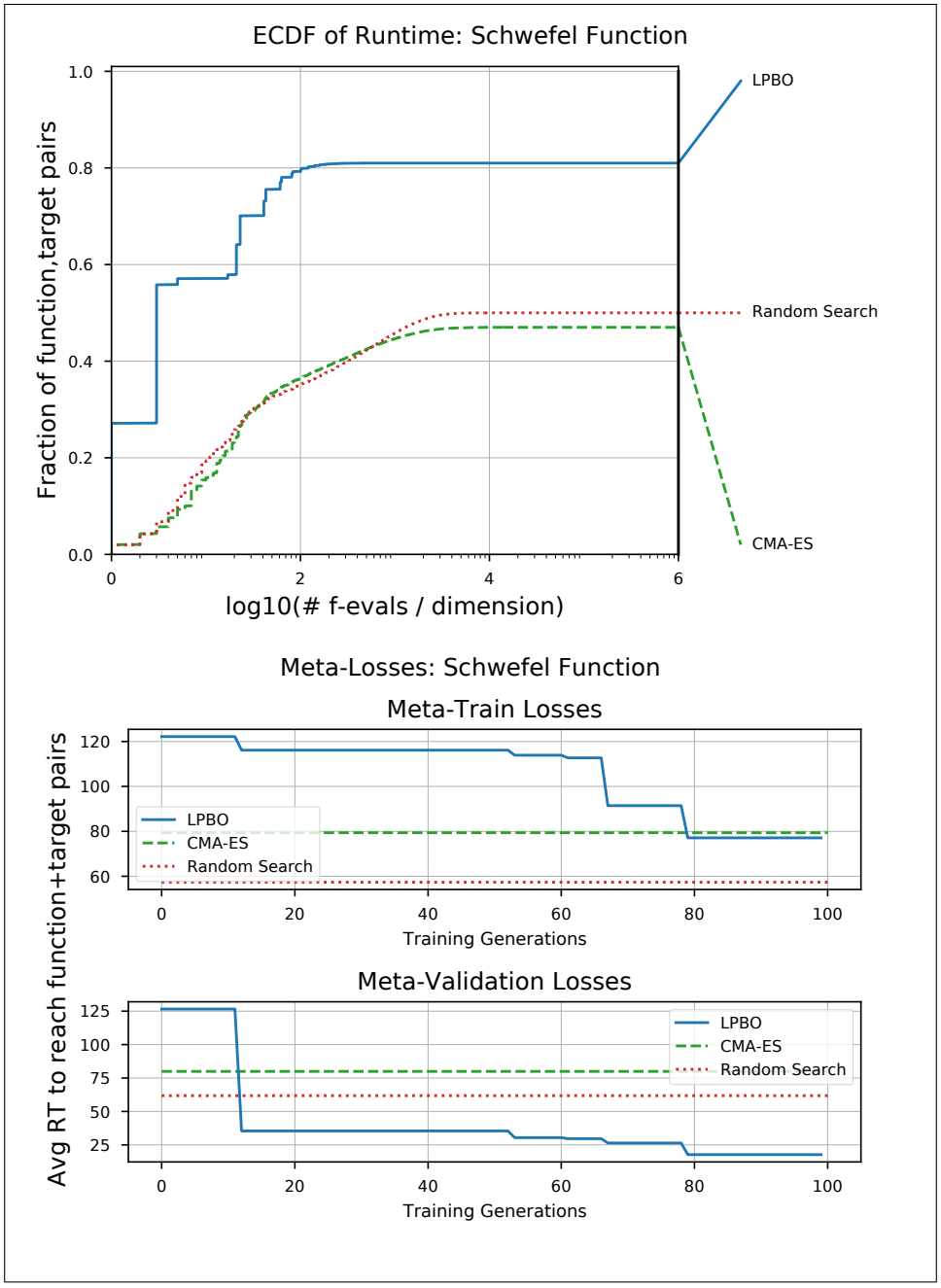

Figure 3.9: Schwefel Function results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

Summary and Discussion

In this section, we have investigated a policy instance's performance in the Learning-To-Optimize Framework for different distributions of black-box optimization tasks. The meta-learning algorithm has shown promising results in most scenarios, while it is not required

fine-tuning in the meta-optimizer and policy hyperparameters.

In particular, the *Linear Slope*, *Lunacek bi-Rastrigin* and *Schwefel Function* present the best LPBO performance in any budget available scenario when compared to all other approaches. We discuss possible strategies to improve the results, such as hybrid approaches, where one can embed some known search mechanisms from other population-based algorithms into the policy; or the initial population, where one can also improve the first population's performance to have a more robust policy for a particular case. Our proposed meta-loss 2.12 have shown to be robust for different cases. The reason is that we leverage sample efficiency (low budget available) and robustness (high budget available) using the conceptual restart algorithm.

3.6 Experimental Analysis II: Group BBO Functions

In this section, we provide the second part of our experiments. It aims to show the generalization of the method in broad classes of optimization problems while discussing its performance in higher dimensions. We study two groups of functions in the meta-training in 2D, 5D, and 10D:

- Group 1 (Multi-modal functions with adequate global structure):
  - Rastrigin Function
  - Weierstrass Function
  - Shaffers F7 Function
  - Shaffers F7 Function, moderately ill-conditioned
  - Composite Griewank-Rosenbrck Function F8F2
- Group 2 (Multi-modal functions with weak global structure):
  - Schwefel Function
  - Gallagher's Gaussian 101-me Peaks Function
  - Gallagher's Gaussian 21-me Peaks Function
  - Katsuura Function
  - Lunacek bi-Rastrigin Function

In experiment analysis II, the meta-training scenario is very different. The optimizer should learn to exploit different properties in the functions while it is also learning to explore. The exploration is essential here due to multiple local optima and the number of different strategies required to exploit the diversity of each group's function.

We chose to keep the same evaluation procedure as described before. We check the performance against the state-of-the-art method CMA-ES and Random Search. We also keep the same hyperparameters for the policy, and the meta-optimizer to show zero (or minimal) fine-tuning might be required for different black-box optimization tasks. However, further investigation is needed to check for higher performance in different scenarios when we change all hyperparameters.

Group 1: Dimension 2D

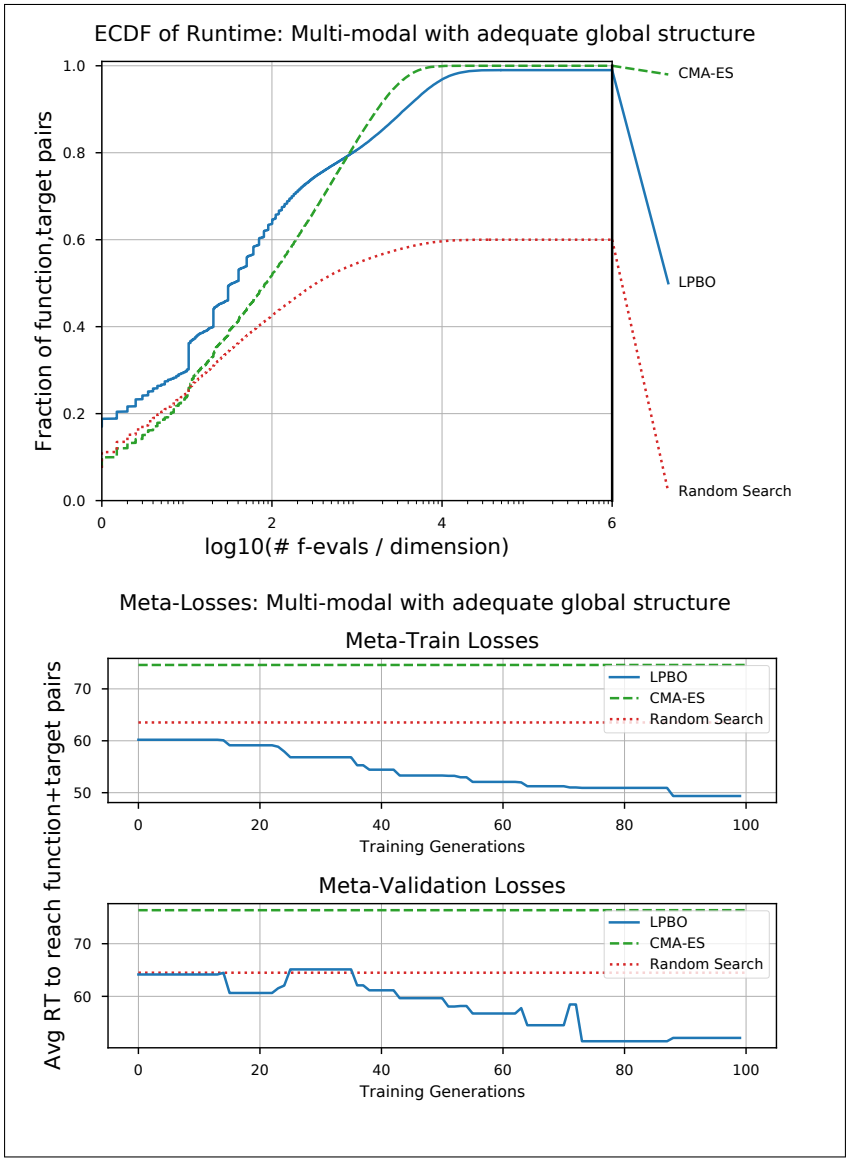

Figure 3.10: Group 1 in 2D results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

Figure 3.10 depicts the results for Group 1. This group of functions has different properties to provide an adequate global structure for the global optimizer. For example, the *Weierstrass Function* has repetitive landscapes while *Rastrigin Function* has a very regular placement of the optimal point. This fact suggests that the optimizer can not rely on simple strategies such as "only sample points in the domain boundary" or "locate the plateau where the optima might be". As a result, we expect the leaned optimizer to develop a better exploration strategy than the learned optimizer in a single function case. This analysis is important to interpret the results correctly.

In the ECDF of runtimes graph, the results of RS are worse than all the other methods. Besides, superior results are reported for the LPBO when considering a lower budget (i.e., when a few function evaluations are available to the solver). Our results demonstrated that the learned algorithm developed a better exploration approach intrinsic to the method's population update. Although CMA-ES has a better result in a small gap in the ECDF graph, the LPBO method can solve all tasks if given more budget. It might suggest a tradeoff between the initialization of the population and the first updates in each generation to solve the most straightforward tasks faster, but it takes more time to escape from the local optima and solve more challenging tasks.

Group 2: Dimension 2

The COCO platform assigns the second group as multi-modal functions with a weak global structure. In the case of *Gallagher's Gaussian 21-hi Peaks* and *Gallagher's Gaussian 101-me Peaks* functions, there are multiple optima with position and height being unrelated and randomly chosen in each instance. In the case of all other functions, they are highly rugged and might have more than $10 * D$ global optimal points. They might also be considered deceptive for evolutionary algorithms with a large population size. All those different features create an enormous challenge to find any suitable structure that can be exploited by the learning algorithm.

Figure 3.11 displays an example that perfectly illustrates what a learned algorithm can improve over a hand-engineered algorithm (CMA-ES or RS). The CMA-ES and RS have similar results, while LPBO gives clearly better results in all scenarios. These results point to a better strategy to learn the initial population. It seems to provide a better optimization path leading to better overall performance. This experiment might be considered the most challenging group in the COCO platform due to its high variability, and the search strategies are usually fine-tuned to work efficiently in those scenarios.

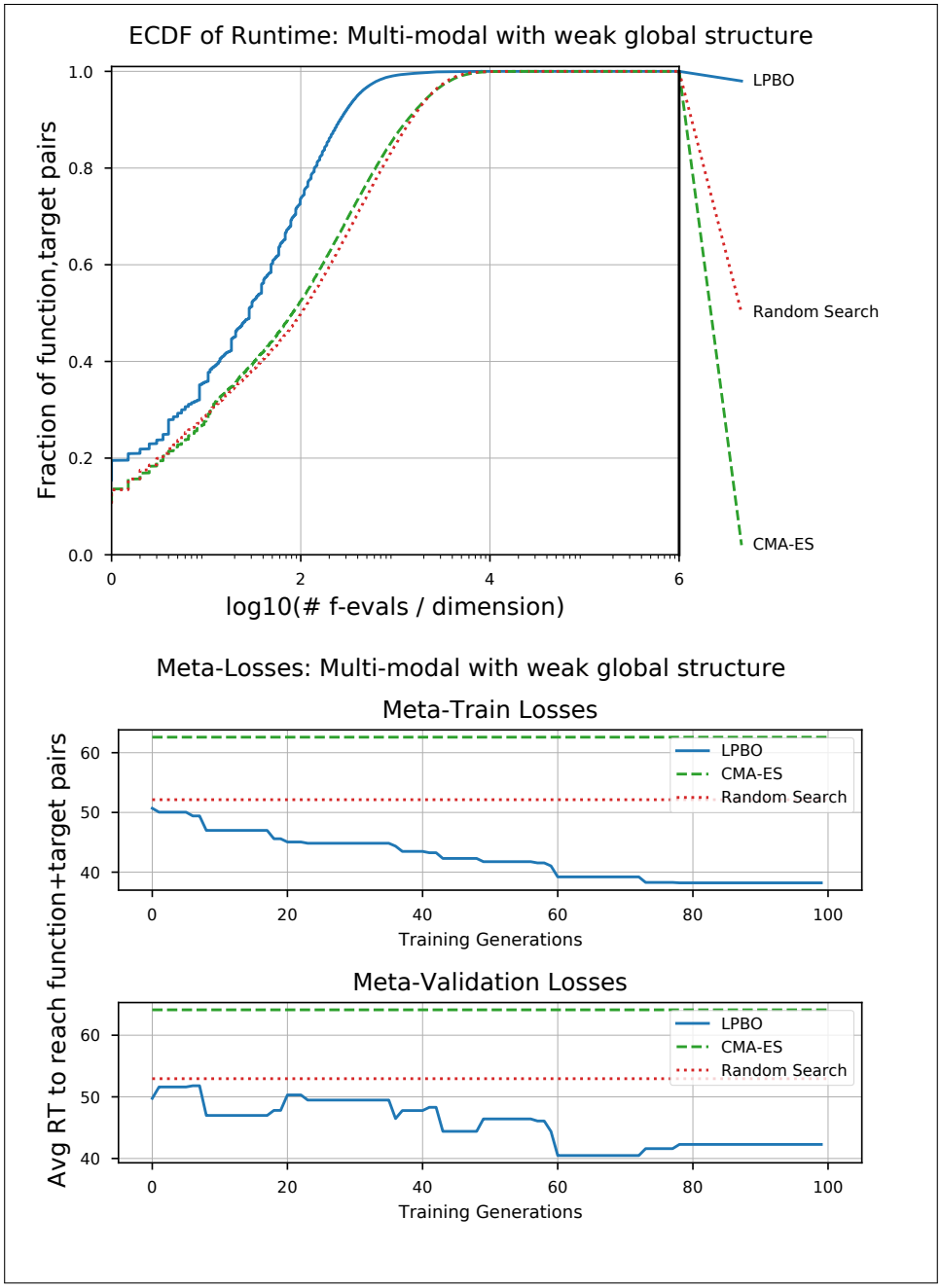

Figure 3.11: Group 2 in 2D results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

Dimensions 5 and 10

In this section, we briefly examine the scalability of the Learning-To-Optimize POMDP in higher dimensions. Figure 3.12 displays the results for Group 1 in 5D and 10D and Figure 3.13 reports the results for Group 2 in 5D and 10D. All used functions remain the same as previously evaluated, the only difference is that we aggregate the results in different dimensions.

Concerning the ECDF of runtimes (Figures 3.12 and 3.13; top), we can observe a slight improvement of the LPBO over CMA-ES and RS. In the group 2, all three methods present similar results regarding their curve tendency and final values. However, in group 1, the LPBO shows a more robust strategy for the current scenario. It is possible to notice the difference between the three methods with the superior performance of the LPBO. Notably, LPBO is able to solve more challenging tasks when we consider a longer run. A further novel finding is that the pattern in the case of group 1 and 10D is that both CMA-ES and LPBO has the same relative difference when compared to Figure 3.10. This can be seen as a bias on the hyperparameters chosen or the task distribution. Further research is needed to confirm this novel finding.

The meta-loss validation (Figures 3.12 and 3.13; bottom) shows more variability in evaluating the policy when compared to the meta-train loss. This instability can be seen for all datasets. Possibly, the size of the dataset influences the training regime of the LPBO. A bigger dataset may reduce the noise and improve the validation evaluation on these conditions. Even so, the validation performance appears to reflect the training performance. They tend to decrease over time. In the following section, we summarize the whole work and explain the contributions and limitations of our approach and conclude with possible future research directions.

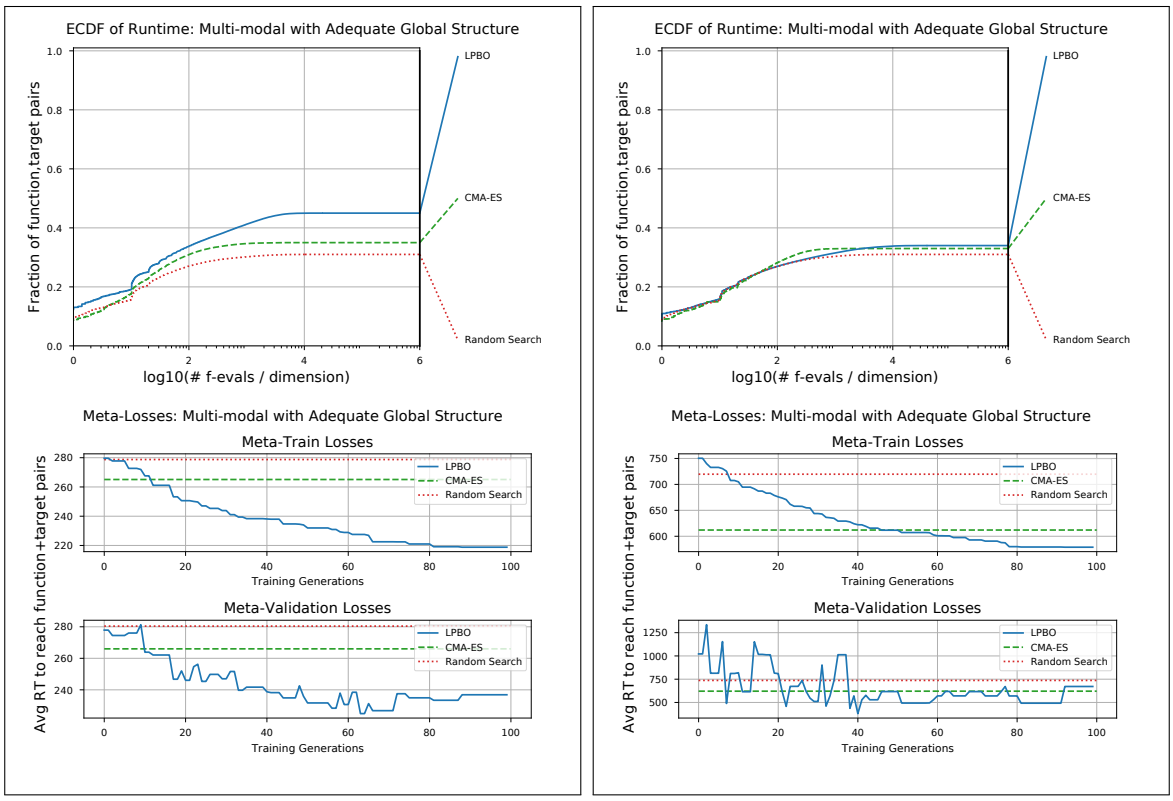

Figure 3.12: Group 1 in 5D (left) and 10D (right) results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

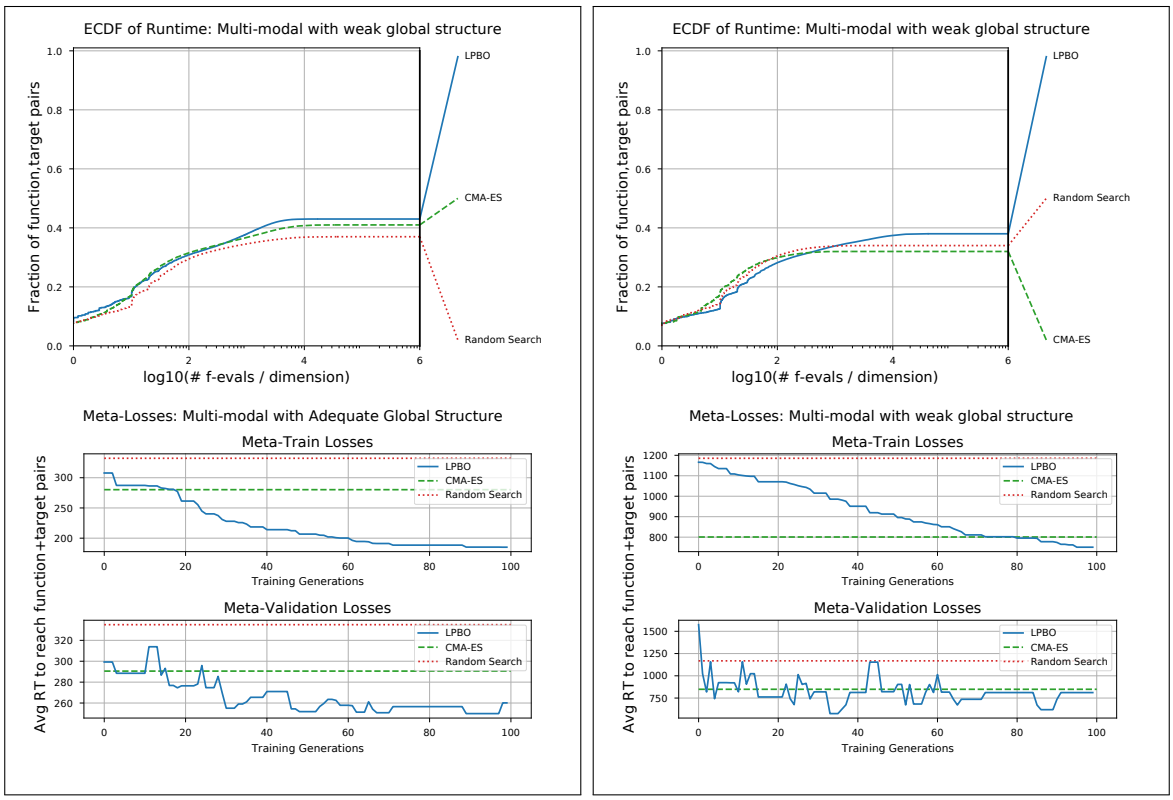

Figure 3.13: Group 2 in 5D (left) and 10D (right) results: Empirical cumulative distribution of simulated runtimes (top); Meta-train and meta-validation losses of the LPBO, CMA-ES and Random Search (bottom)

3.7 Conclusion

This chapter presented the results obtained by learning to learn a population-based optimizer with the learning-to-optimize POMDP implementation. In conclusion, we have observed a considerable decrease in the average number of evaluations required to achieve success criteria in various scenarios. The most noticeable results were in functions that have an adequate global structure. This finding suggests that the LPBO can indeed exploit general structures while exploring promising regions. In the case *Schwefel Function*, *Ellipsoidal Functions*, *Lunacek bi-Rastrigin Function*, and *Linear Slope*, our model was shown to be around two times faster than the state-of-the-art CMA-ES optimizer. Even so, it could solve more targets (i.e., be more explorative) than Random Search. A strong feature of our method is the potential to achieve a good trade-off between precision and speed.

Also, it was found that the initial population was a significant factor in our algorithm's performance. A simple and straightforward example of this phenomena can be seen in the *Linear Slope*. We know beforehand that the optimal points will be in the domain boundaries. Therefore the algorithm does not need to explore the function's inner regions and can have an outstanding performance on the first function evaluations (Figure 3.3).

The method in LPBO was more successful in achieving better targets than CMA-ES and Random Search in higher dimensions. However, its performance deteriorated with dimensions increase. This decrease is expected in general cases, but we believe that the policy architecture's LSTM module plays an important role in this case. Since the hyperparameters were fixed, the architecture's memory capacity remained the same over all experiments. Further investigation on the representation of the policy might increase the adaptability for more diverse and complex datasets.

Conclusion

The dominant approach in the optimization community is either to propose hand-engineered search mechanisms or to use generic ones; instead, we believe that another path may be more successful at generating efficient general-purpose population-based optimizers. In this work, we investigated whether or not learnable population-based algorithms were a feasible way to achieve improved results relative to state-of-the-art algorithms, such as CMA-ES. To evaluate this, we planned and developed Learning to optimize POMDP, a general meta-learning approach for black-box optimization problems.

Initially, we studied the adaptability of population-based optimizers and described mostly search mechanisms as a common structure. We were able to identify a general structure of population-based algorithms that support the understanding of a learned population-based optimizer dynamics. In summary, this study argued that different hand-engineered rules can be formalized as a general mapping function from a set of evaluations of the past points to a new set of points. Although this idea is simple, it supports one of the initial hypothesis of this work. It was found that efforts to identify an effective search mechanism could be reshaped as a learning problem. Thus, the general mapping function can be parameterized and optimized, given some metric (i.e., loss function), to express population-based algorithms.

Towards this point, we introduce a POMDP-based meta-learning framework as a general methodology to learn population-based optimizers. The key purpose of the *learning to optimize POMDP* framework is to allow the training of learnable population-based optimizers. By assuming no additional information about the training functions, except their raw evaluations, the framework was then utilized to create a learned population-based optimizer (LPBO) from scratch. The evolved algorithm were tested and compared with a state-of-the-art algorithm (CMA-ES), and Random Search. The algorithm was able to improve the established baseline in most cases. In some complex scenarios, such as the Lunacek bi-Rastrigin Function or Schwefel, the LPBO solved more tasks while being more sample efficient at solving them. It might suggest that learning algorithms could exceed hand-engineered algorithms' performance, currently used in different areas. Furthermore, the LPBO has also shown similar performance to CMA-ES in higher dimensions, suggesting that the evolved policy might have implicitly discovered similar population update mechanisms.

Additionally, this work demonstrated the viability of the approach via the proposed framework. All its elements provide a potential mechanism for generating new search algorithms while standardizing a correct evaluation process of those methods. We found that the learning algorithms' requirements, such as the policy architecture or the meta-optimizer, are surprisingly reasonable and practical. When we trained a simple LSTM policy with Bayesian stochastic weights while using Deep GA with minor modifications, we could outmatch CMA-ES in various scenarios. In the future, we would like to investigate the representation of more sophisticated neural networks or meta-optimizer methods in other scenarios.

In conclusion, the holy grail of black-box optimization might not be a single optimization algorithm but a learning algorithm that can adapt for any optimization task given the distribution of problems. In the same path that recent machine learning approaches have shown promising results by shifting from hand-engineered features to learned features, we believe that optimization should follow the same path.

Contributions

- **Development of the Learning-To-Optimize POMDP:** The main contribution of this work was to provide an approach to learn population-based algorithms. In this work, we formalized *learning to optimize* as a policy search problem on a particular POMDP, which we refered to as *Learning-To-Optimize POMDP*. This formalization allows a BBO practitioner to learn and evaluate general-purpose optimizers given a specific context. Besides, we made a sensitivity analysis to understand the impact of a meta-learning approach on population-based optimizers regarding adaptability and performance.

- **The adoption of benchmarking black-box optimization tasks as a dataset to train population-based optimizers:** Recent works in learning to optimize have proposed to train optimizers mostly using gradient-based meta-losses. Those functions usually require features computed during training or validation. In this work, we found that learn to optimize from scratch without using any additional feature except the evaluation of each point is feasible and improve results over established baselines.

- **Learned Population-based Algorithms as an alternative to hand-engineered methods:** By learning a parameterized population-based optimizer, we achieved strong results over a wide variety of problems. In particular, we found that the proposed meta-loss training function and the policy architecture's invariance property significantly improved the results. We evaluated a parameterized optimizer's performance in two scenarios: multiple instances of a black-box function and a group of related functions; and showed that the approach was more efficient in most scenarios against CMA-ES and Random Search.

- **Robustness of EAs in training learnable optimizers in a meta-learning setting:** There are numerous applications where one can apply evolutionary algorithms. Recent works increased the interest in applying those algorithms for machine learning tasks in generating new computer vision architectures [92] as well as reinforcement learning tasks [21]. Following this trend, we found in this thesis that genetic algorithms can be robust and efficient to train learnable optimizers with a fix set of hyperparameters for many scenarios.

This work opens up several avenues for future research.

- **Deeper levels of meta-learned optimizers:** Although this work introduces a method to learn population-based algorithms, further investigation of the method's variants might be required. The influence of the policy architecture and the meta-optimizer might change the search behavior of the learned algorithm. Another influence can appear in the choice of the training optimization tasks. In summary, future work could be done to also learn to learn those hyperparameters of the proposed approach. It introduces the notion of multiple levels of meta-learning optimization algorithms.
- **Different modalities in learning black-box optimization:** We have limited this work to *continuous unconstrained black-box optimization*. We believe the framework can be easily adjusted to other cases such as *constrained* or *discrete* optimization.
- **Learning to chose tasks:** We have limited this work to a fixed set of tasks. In other words, the performance of the learned optimizer depends on the tasks available for meta-training. However, we believe that learn how to select the best task design might be beneficial to the performance of the learned algorithm.

Bibliography

[1] M. Claesen, J. Simm, D. Popovic, and B. D. Moor, "Hyperparameter tuning in Python using Optunity," in *Proceedings of the International Workshop on Technical Computing for Machine Learning and Mathematical Engineering*, vol. 1, p. 3, 2014.

[2] K. Chatzilygeroudis, R. Rama, R. Kaushik, D. Goepp, V. Vassiliades, and J.-B. Mouret, "Black-box data-efficient policy search for robotics," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 51–58, IEEE, 2017.

[3] E. Cabrera, M. Taboada, M. L. Iglesias, F. Epelde, and E. Luque, "Optimization of healthcare emergency departments by agent-based simulation," *Procedia computer science*, vol. 4, pp. 1880–1889, 2011. Publisher: Elsevier.

[4] B. Liu, L. Wang, Y. Liu, and S. Wang, "A unified framework for population-based metaheuristics," *Annals of Operations Research*, vol. 186, no. 1, pp. 231–262, 2011. Publisher: Springer.

[5] K. Li and J. Malik, "Learning to optimize," *arXiv preprint arXiv:1606.01885*, 2016.

[6] F. Wang, "Measurement, Optimization, and Impact of Health Care Accessibility: A Methodological Review," *Annals of the Association of American Geographers. Association of American Geographers*, vol. 102, pp. 1104–1112, Sept. 2012.

[7] Z. Bouzarkouna, D. Y. Ding, and A. Auger, "Using evolution strategy with meta-models for well placement optimization," in *ECMOR XII-12th European Conference on the Mathematics of Oil Recovery*, pp. cp–163, European Association of Geoscientists & Engineers, 2010.

[8] N. Qayyum, A. Amin, U. Jamil, and A. Mahmood, "Optimization techniques for home energy management: A review," in *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pp. 1–7, IEEE, 2019.

[9] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[10] N. Andreasson, M. Patriksson, and A. Evgrafov, *An introduction to continuous optimization: foundations and fundamental algorithms*. Courier Dover Publications, 2020.

[11] F. Werner, *Advances and Novel Approaches in Discrete Optimization*. Multidisciplinary Digital Publishing Institute, 2020.

[12] J. T. Linderoth and M. W. Savelsbergh, "A computational study of search strategies for mixed integer programming," *INFORMS Journal on Computing*, vol. 11, no. 2, pp. 173–187, 1999. Publisher: INFORMS.

[13] J.-H. Kim and H. Myung, "Evolutionary programming techniques for constrained optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 2, pp. 129–140, 1997. Publisher: IEEE.

[14] J. E. Dennis Jr and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.

[15] K. Kawaguchi, J. Huang, and L. P. Kaelbling, "Every Local Minimum Value is the Global Minimum Value of Induced Model in Non-convex Machine Learning," *Neural Computation*, vol. 31, pp. 2293–2323, Dec. 2019. arXiv: 1904.03673.

[16] L. Xiujuan and S. Zhongke, "Overview of multi-objective optimization methods," *Journal of Systems Engineering and Electronics*, vol. 15, no. 2, pp. 142–146, 2004. Publisher: BIAI.

[17] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[18] K. De Jong, "Evolutionary computation: a unified approach," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pp. 185–199, 2016.

[19] T. Gabor, L. Belzner, T. Phan, and K. Schmid, "Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms," in *2018 IEEE International Conference on Autonomic Computing (ICAC)*, pp. 131–140, IEEE, 2018.

[20] E.-G. Talbi, *Metaheuristics: from design to implementation*. Hoboken, N.J: John Wiley & Sons, 2009. OCLC: ocn230183356.

[21] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," *arXiv:1703.03864 [cs, stat]*, Sept. 2017. arXiv: 1703.03864.

[22] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[23] T. Bartz-Beielstein, C. Doerr, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, M. Lopez-Ibanez, K. M. Malan, J. H. Moore, *et al.*, "Benchmarking in optimization: Best practice and open issues," *arXiv preprint arXiv:2007.03488*, 2020.

[24] R. Shang, Y. Wang, J. Wang, L. Jiao, S. Wang, and L. Qi, "A multi-population cooperative coevolutionary algorithm for multi-objective capacitated arc routing problem," *Information Sciences*, vol. 277, pp. 609–642, 2014. Publisher: Elsevier.

[25] B. T. Tesfaldet and A. Y. Hermosilla, "A Lamarckian genetic algorithm applied to the travelling salesman problem," *Advances in Complex Systems*, vol. 2, no. 04, pp. 431–457, 1999. Publisher: World Scientific.

[26] C. Fernando, J. Sygnowski, S. Osindero, J. Wang, T. Schaul, D. Teplyashin, P. Sprechmann, A. Pritzel, and A. Rusu, "Meta-learning by the baldwin effect," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 1313–1320, 2018.

[27] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, Reading: MA, 1989.

[28] K. A. De Jong, "Genetic algorithms: A 10 year perspective," in *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, vol. 1, pp. 9–177, Lawrence Erlbaum Associates, 1985. Issue: 6.

[29] E. Alba, F. Luna, A. J. Nebro, and J. M. Troya, "Parallel heterogeneous genetic algorithms for continuous optimization," *Parallel Computing*, vol. 30, no. 5-6, pp. 699–719, 2004. Publisher: Elsevier.

[30] P. Esfahanian and M. Akhavan, "GACNN: Training Deep Convolutional Neural Networks with Genetic Algorithm," *arXiv preprint arXiv:1909.13354*, 2019.

[31] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[32] J. Klockgether and H.-P. Schwefel, "Two-phase nozzle and hollow core jet experiments," in *Proc. 11th Symp. Engineering Aspects of Magnetohydrodynamics*, pp. 141–148, Pasadena, CA: California Institute of Technology, 1970.

[33] H.-G. Beyer, *The theory of evolution strategies*. Springer Science & Business Media, 2001.

[34] N. Hansen, "The CMA evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[35] N. Müller and T. Glasmachers, "Challenges in high-dimensional reinforcement learning with evolution strategies," in *International Conference on Parallel Problem Solving from Nature*, pp. 411–423, Springer, 2018.

[36] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Natural evolution strategies," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 3381–3387, IEEE, 2008.

[37] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, *Towards a new evolutionary computation: advances on estimation of distribution algorithms*, vol. 192. Springer, 2006.

[38] H. Mühlenbein and G. Paass, "From recombination of genes to the estimation of distributions I. Binary parameters," in *International conference on parallel problem solving from nature*, pp. 178–187, Springer, 1996.

[39] C. González, J. A. Lozano, and P. Larranaga, "The convergence behavior of the PBIL algorithm: a preliminary approach," in *Artificial Neural Nets and Genetic Algorithms*, pp. 228–231, Springer, 2001.

[40] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proceedings of the genetic and evolutionary computation conference GECCO-99*, vol. 1, pp. 525–532, Citeseer, 1999.

[41] K. D. Boese, A. B. Kahng, and S. Muddu, "A new adaptive multi-start technique for combinatorial global optimizations," *Operations Research Letters*, vol. 16, no. 2, pp. 101–113, 1994. Publisher: Elsevier.

[42] P. Pinto, T. A. Runkler, and J. M. Sousa, "Wasp swarm optimization of logistic systems," in *Adaptive and Natural Computing Algorithms*, pp. 264–267, Springer, 2005.

[43] K. C. Steer, A. Wirth, and S. K. Halgamuge, "The rationale behind seeking inspiration from nature," in *Nature-Inspired Algorithms for Optimisation*, pp. 51–76, Springer, 2009.

[44] A. Tzanetos and G. Dounias, "A Comprehensive Survey on the Applications of Swarm Intelligence and Bio-Inspired Evolutionary Strategies," in *Machine Learning Paradigms: Advances in Deep Learning-based Technological Applications* (G. A. Tsihrintzis and L. C. Jain, eds.), pp. 337–378, Cham: Springer International Publishing, 2020. Type: 10.1007/978-3-030-49724-8_15.

[45] M. Dorigo, "Optimization, learning and natural algorithms [Ph. D. thesis]," *Politecnico di Milano, Italy*, 1992.

[46] M. Dorigo and T. Stützle, "The ant colony optimization metaheuristic: Algorithms, applications, and advances," in *Handbook of metaheuristics*, pp. 250–285, Springer, 2003.

[47] X. Dai, S. Long, Z. Zhang, and D. Gong, "Mobile robot path planning based on ant colony algorithm with A* heuristic method," *Frontiers in neurorobotics*, vol. 13, p. 15, 2019. Publisher: Frontiers.

[48] K. Socha, "ACO for continuous and mixed-variable optimization," in *International Workshop on Ant Colony Optimization and Swarm Intelligence*, pp. 25–36, Springer, 2004.

[49] G. Leguizamón and C. A. C. Coello, "An Alternative ACO $ _ {\Bbb {R}} $ Algorithm for Continuous Optimization Problems," in *International Conference on Swarm Intelligence*, pp. 48–59, Springer, 2010.

[50] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, pp. 1155–1173, Mar. 2008.

[51] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, vol. 4, pp. 1942–1948, Citeseer, 1995.

[52] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, pp. 649–657, 2015.

[53] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, and A. Ray, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020. Publisher: SAGE Publications Sage UK: London, England.

[54] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, "Deep learning techniques for music generation–a survey," *arXiv preprint arXiv:1709.01620*, 2017.

[55] D. Güera and E. J. Delp, "Deepfake video detection using recurrent neural networks," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, IEEE, 2018.

[56] C. Finn, K. Xu, and S. Levine, "Probabilistic model-agnostic meta-learning," in *Advances in Neural Information Processing Systems*, pp. 9516–9527, 2018.

[57] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in neural information processing systems*, pp. 4077–4087, 2017.

[58] J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess, "Meta reinforcement learning as task inference," *arXiv preprint arXiv:1905.06424*, 2019.

[59] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997. Publisher: Springer.

[60] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Advances in Neural Information Processing Systems*, pp. 350–360, 2019.

[61] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *ICLR*, 2017.

[62] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," *arXiv preprint arXiv:1806.04910*, 2018.

[63] L. Metz, N. Maheswaranathan, C. D. Freeman, B. Poole, and J. Sohl-Dickstein, "Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves," *arXiv preprint arXiv:2009.11243*, 2020.

[64] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei, "On the optimization of a synaptic learning rule," in *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, vol. 2, Univ. of Texas, 1992.

[65] T. P. Runarsson and M. T. Jonsson, "Evolution and design of distributed learning rules," in *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks. Proceedings of the First IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks (Cat. No. 00*, pp. 59–63, IEEE, 2000.

[66] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in neural information processing systems*, pp. 3981–3989, 2016.

[67] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8624–8628, IEEE, 2013.

[68] X. Chen, S. Z. Wu, and M. Hong, "Understanding gradient clipping in private SGD: A geometric perspective," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[69] K. Lv, S. Jiang, and J. Li, "Learning gradient descent: Better generalization and longer horizons," *arXiv preprint arXiv:1703.03633*, 2017.

[70] C. Daniel, J. Taylor, and S. Nowozin, "Learning Step Size Controllers for Robust Neural Network Training.," in *AAAI*, pp. 1519–1525, 2016.

[71] C. Xu, T. Qin, G. Wang, and T.-Y. Liu, "Reinforcement learning for learning rate control," *arXiv preprint arXiv:1705.11159*, 2017.

[72] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," *arXiv preprint arXiv:1709.07417*, 2017.

[73] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," *arXiv preprint arXiv:1703.04813*, 2017.

[74] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein, "Understanding and correcting pathologies in the training of learned optimizers," in *International Conference on Machine Learning*, pp. 4556–4565, 2019.

[75] L. Metz, N. Maheswaranathan, R. Sun, C. D. Freeman, B. Poole, and J. Sohl-Dickstein, "Using a thousand optimization tasks to learn hyperparameter search strategies," *arXiv preprint arXiv:2002.11887*, 2020.

[76] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff, "COCO: A platform for comparing continuous optimizers in a black-box setting," *Optimization Methods and Software*, pp. 1–31, 2020. Publisher: Taylor & Francis.

[77] Z. Xu, A. M. Dai, J. Kemp, and L. Metz, "Learning an Adaptive Learning Rate Schedule," *arXiv preprint arXiv:1909.09712*, 2019.

[78] J. Peters, K. Mülling, and Y. Altun, "Relative entropy policy search.," in *AAAI*, vol. 10, pp. 1607–1612, Atlanta, 2010.

[79] N. Maheswaranathan, L. Metz, G. Tucker, D. Choi, and J. Sohl-Dickstein, "Guided evolutionary strategies: Augmenting random search with surrogate gradients," in *International Conference on Machine Learning*, pp. 4264–4273, PMLR, 2019.

[80] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[81] R. Kaplow, A. Atrash, and J. Pineau, "Variable resolution decomposition for robotic navigation under a POMDP framework," in *2010 IEEE International Conference on Robotics and Automation*, pp. 369–376, IEEE, 2010.

[82] T. B. Wolf and M. J. Kochenderfer, "Aircraft collision avoidance using Monte Carlo real-time belief space search," *Journal of Intelligent & Robotic Systems*, vol. 64, no. 2, pp. 277–298, 2011. Publisher: Springer.

[83] B. Wahlberg, "Partially observed Markov decision processes: from filtering to controlled sensing [bookshelf]," *IEEE Control Systems Magazine*, vol. 39, no. 4, pp. 76–79, 2019. Publisher: IEEE.

[84] M. L. Littman, "A tutorial on partially observable Markov decision processes," *Journal of Mathematical Psychology*, vol. 53, no. 3, pp. 119–125, 2009. Publisher: Elsevier.

[85] Y. Cao, T. Chen, Z. Wang, and Y. Shen, "Learning to optimize in swarms," in *Advances in Neural Information Processing Systems*, pp. 15044–15054, 2019.

[86] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep

reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015. Publisher: Nature Publishing Group.

[87] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl $^ 2$: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[88] Y. Ollivier, L. Arnold, A. Auger, and N. Hansen, "Information-geometric optimization algorithms: A unifying picture via invariance principles," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 564–628, 2017. Publisher: JMLR. org.

[89] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[90] J. J. Moré and S. M. Wild, "Benchmarking derivative-free optimization algorithms," *SIAM Journal on Optimization*, vol. 20, no. 1, pp. 172–191, 2009. Publisher: SIAM.

[91] N. Hansen and S. Kern, "Evaluating the CMA evolution strategy on multimodal test functions," in *International Conference on Parallel Problem Solving from Nature*, pp. 282–291, Springer, 2004.

[92] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE transactions on Neural Networks*, vol. 5, no. 1, pp. 54–65, 1994. Publisher: IEEE.